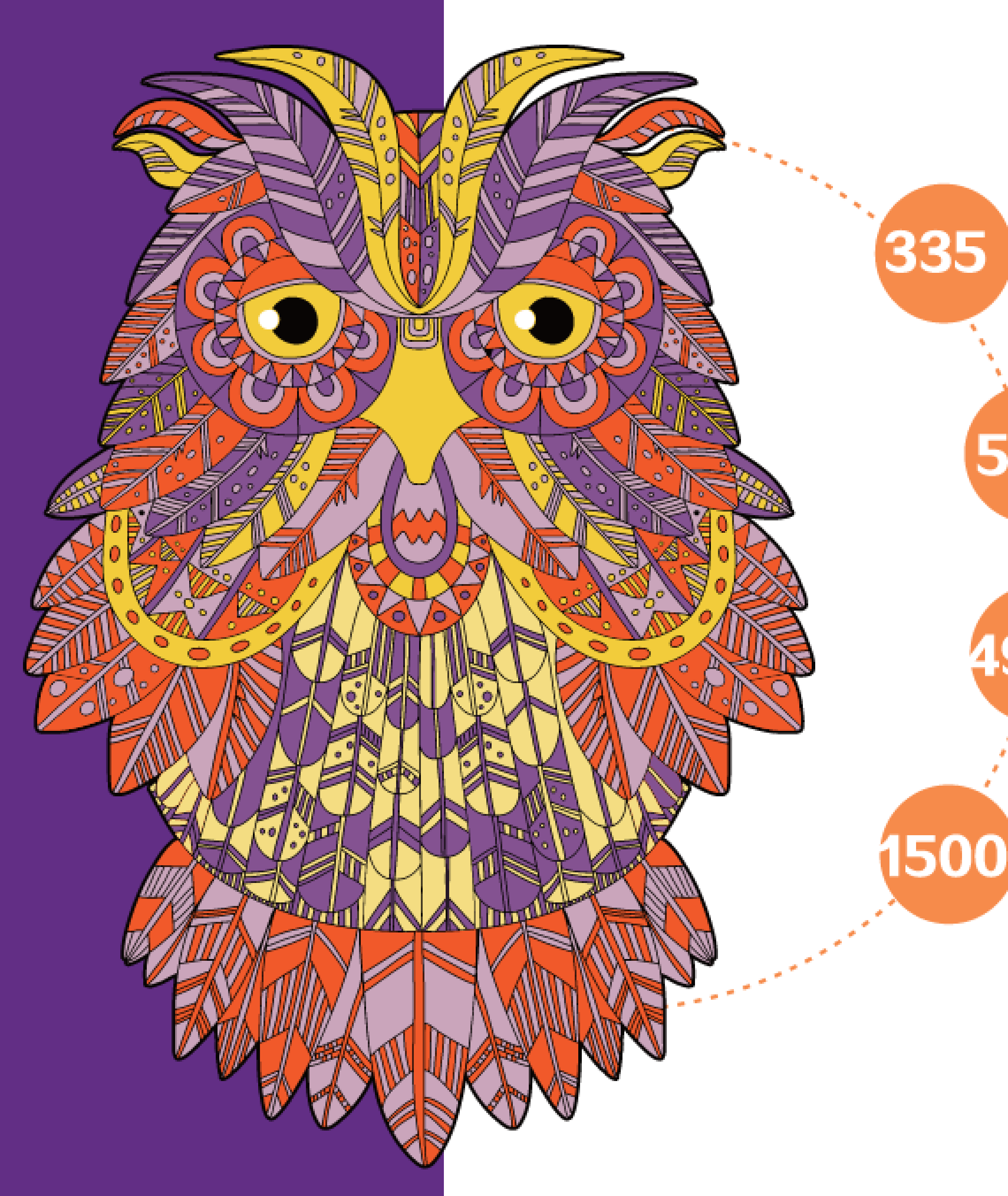
清华社"视频大讲堂"大系
网络开发视频大讲堂



335 节高清微视频 + 188页拓展微阅读 + 271项在线微练习
移动端/PC端同步学习，QQ群/微信群随时答疑。

550 个实例案例分析+338项实例源代码
速查、高效、实用，增强实战能力。

4900 个前端案例 + 48本参考手册
先观摩，再临摹，高手案头常备，随时查阅提升。

1500 套网页模板 + 12000个设计素材 + 1036道前端面试真题
随用随取，提升设计效率，快速进阶开发高手行列。

HTML5+CSS3 从入门到精通

微课精编版

前端科技◎编著

5大类线上资源

- 教学微视频（335节）
- 拓展微阅读（188页）
- 在线微练习（271项）
- 示例效果 (84个)
- 权威参考 (18个)

海量资源，可查可用

- 前端案例资源库（4900个）
- 面试题库（1036道）
- 网页模板库（1500套）
- 设计素材库（12000个）

清华大学出版社



HTML5+CSS3 从入门到精通（微课精编版）

前端科技　编著

清华大学出版社

北　京

内 容 简 介

《HTML5+CSS3 从入门到精通（微课精编版）》从初学者角度出发，通过通俗易懂的语言、丰富多彩的实例，详细介绍了 HTML5+CSS3 前端开发技术及其应用。本书共 25 章，包括 HTML5 基础、HTML5 新增元素和文档结构、HTML5 表单、HTML5 绘图和动画、HTML5 音频和视频、数据存储、应用程序缓存、多线程编程、位置信息、历史记录、文件操作、HTML5 通信、WebRTC 视频直播、跨窗口操作、拖放操作、异步交互、延迟处理、HTML5 其他 API、CSS3 基础、CSS3 文本样式、CSS3 背景图像和渐变背景、CSS3 用户接口样式、CSS3 伸缩盒布局、CSS3 动画、CSS3 媒体查询等内容。书中所有知识都结合具体实例进行介绍，代码注释详尽，读者可轻松掌握前端技术精髓，提升实际开发能力。

除纸质内容外，本书还配备了多样化、全方位的学习资源，主要内容如下：

- ☑ 335节同步教学微视频
- ☑ 15000项设计素材资源
- ☑ 188页拓展知识微阅读
- ☑ 4800个前端开发案例
- ☑ 550个实例案例分析
- ☑ 48本权威参考学习手册
- ☑ 271个在线微练习
- ☑ 1036道企业面试真题

本书可作为前端开发、移动开发入门者的自学用书，也可作为高等院校及相关培训机构的教学参考用书。

图书在版编目（CIP）数据

HTML5+CSS3 从入门到精通：微课精编版 / 前端科技编著. — 北京：清华大学出版社，2018
（清华社“视频大讲堂”大系 网络开发视频大讲堂）
ISBN 978-7-302-50253-1

Ⅰ. ①H…　Ⅱ. ①前…　Ⅲ. ①超文本标记语言—程序设计 ②网页制作工具　Ⅳ. ①TP312.8 ②TP393.092.2

中国版本图书馆 CIP 数据核字（2018）第 114828 号

责任编辑：贾小红
封面设计：李志伟
版式设计：周春梅
责任校对：赵丽杰
责任印制：

出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
　　社 总 机：010-62770175　　**邮　　购：**010-62786544
　　投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
　　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　　张：**35.25　　**字　　数：**1040 千字
版　　次：2018 年 8 月第 1 版　　**印　　次：**2018 年 8 月第 1 次印刷
定　　价：79.80 元

产品编号：078946-01

如何使用本书

本书提供了多样化、全方位的学习资源，帮助读者轻松掌握 HTML5+CSS3 技术，从小白快速成长为前端开发高手。



手机端+PC 端，线上线下同步学习

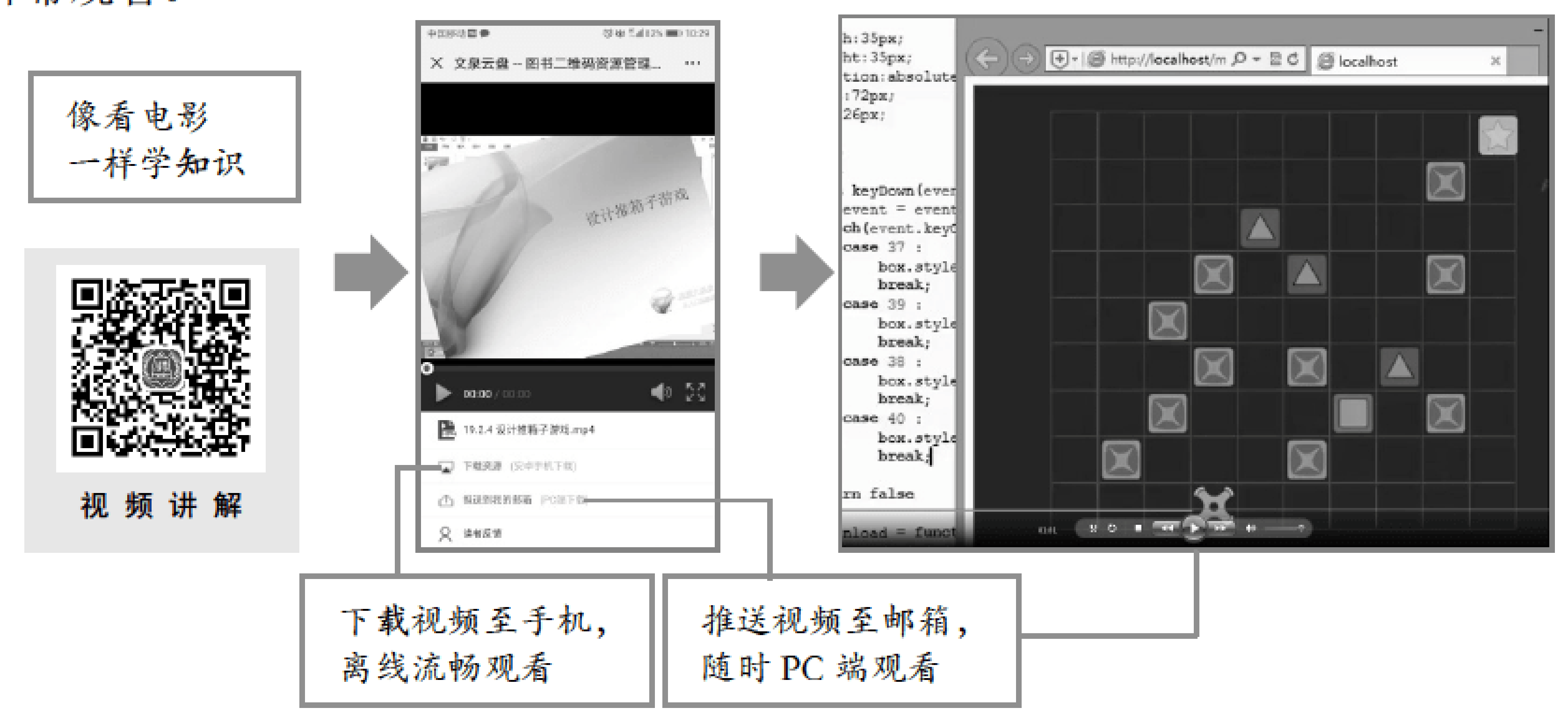
1. 获取学习权限

学习本书前，请先刮开图书封底的二维码涂层，使用手机扫描，即可解锁本书资源的学习权限。再扫描正文章节对应的 5 类二维码，可以观看视频讲解，阅读线上资源，体验示例效果，查阅权威参考资料和在线练习提升，全程易懂、好学、速查、高效、实用。

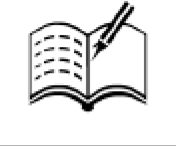


2. 观看视频讲解

对于初学者来说，精彩的知识讲解和透彻的实例解析能够引导其快速入门，轻松理解和掌握知识要点。本书中几乎所有案例都录制了视频，可以使用手机在线观看，也可以离线观看，还可以推送到电脑上大屏幕观看。



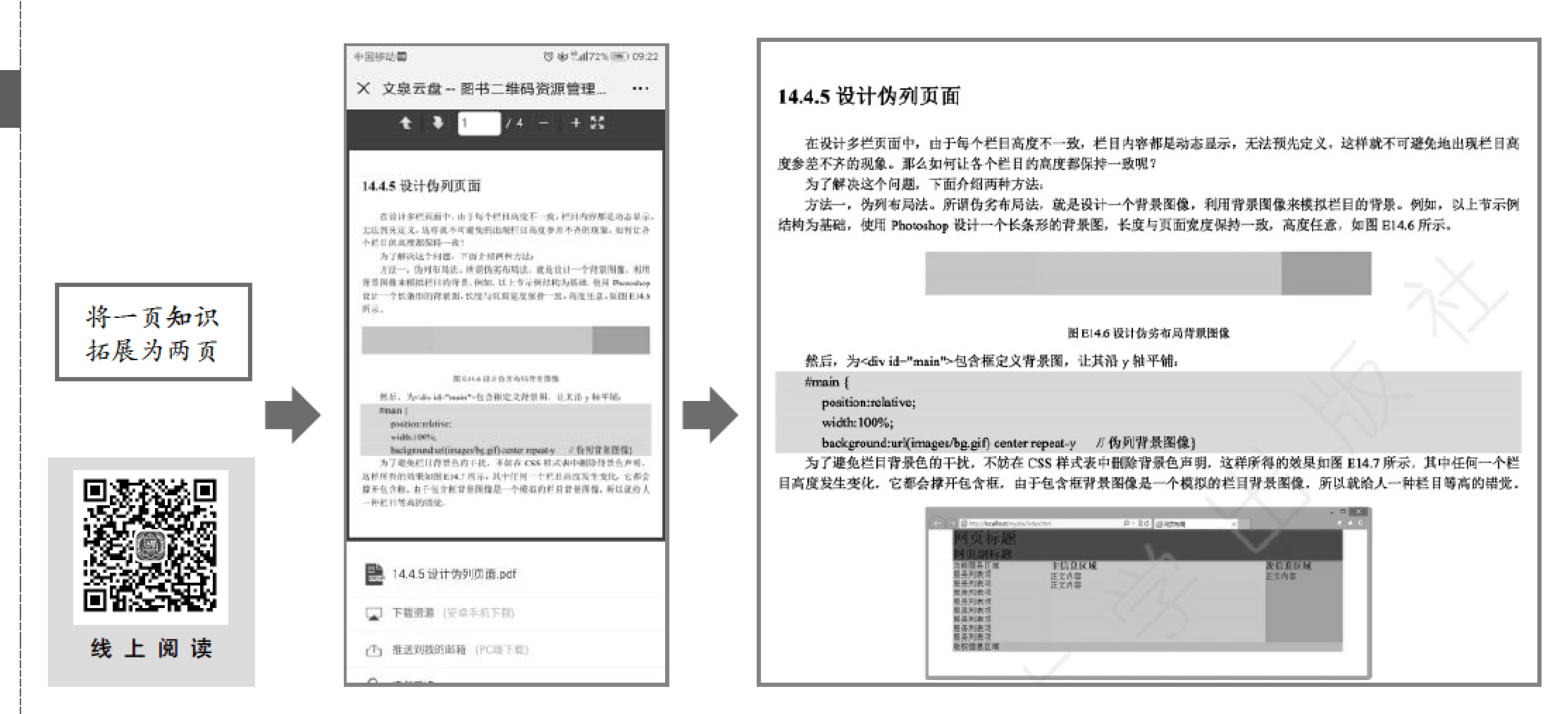






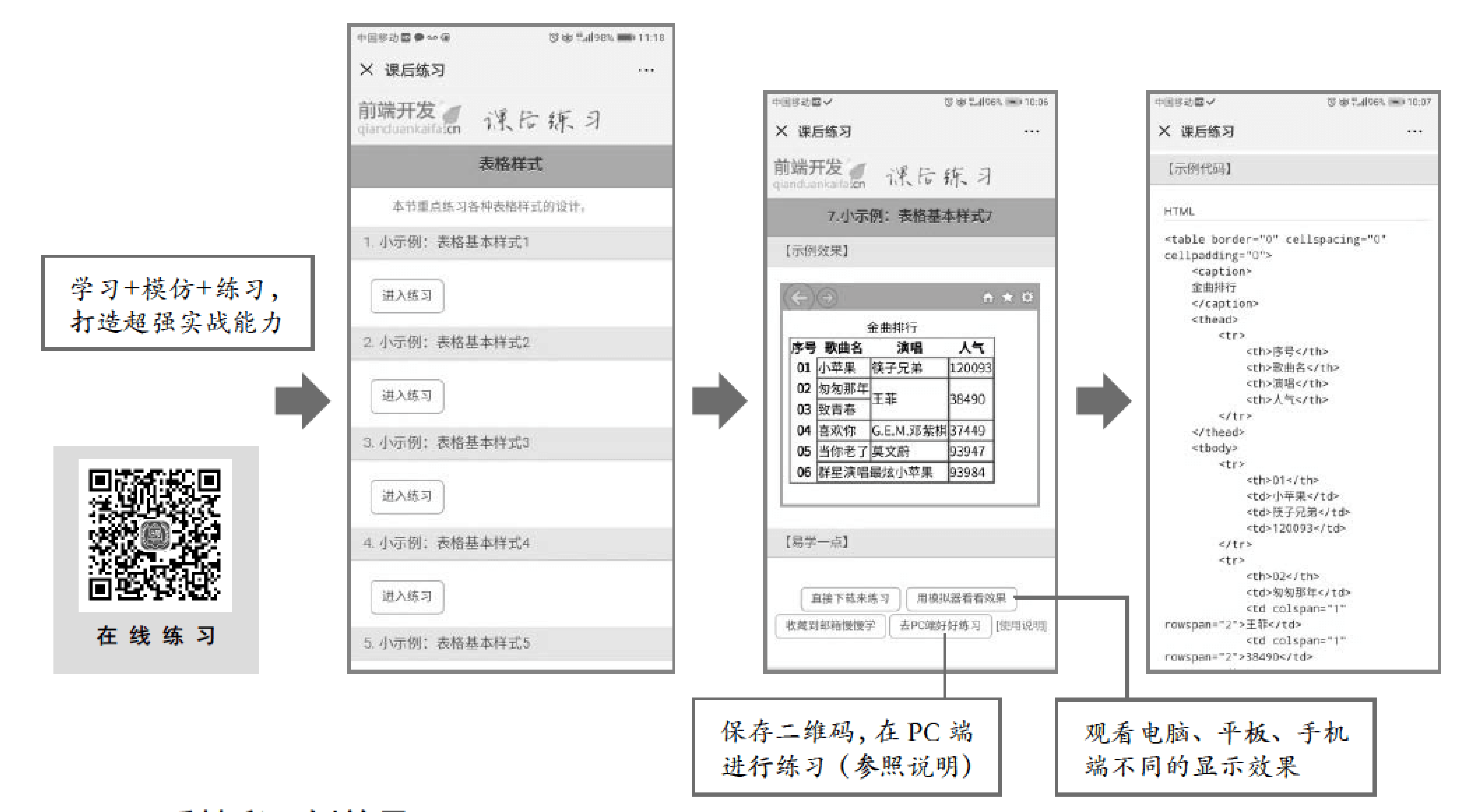
3. 拓展线上阅读

一本书的厚度有限，但掌握一门技术却需要大量的知识积累。本书选择了那些与学习、就业关系紧密的核心知识点印在书中，而将大量的拓展性知识放在云盘上，读者扫描“线上阅读”二维码，即可免费阅读数百页的前端开发学习资料，获取大量的额外知识。



4. 进行线上练习

为方便读者巩固基础知识，提升实战能力，本书附赠了大量的前端练习题目。读者扫描各章最后的“在线练习”二维码，即可通过反复的实操训练加深对知识的领悟程度。

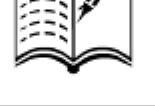


5. 观看精彩示例效果

对于前端设计、开发来说，很多案例效果在纸质书上是无法得到完美呈现的，如动态效果、绚丽页面等。因此本书特意提供了部分示例效果展示。读者扫描案例旁的“示例效果”二维码，即可在学习过程中直观感受到精彩的页面效果。

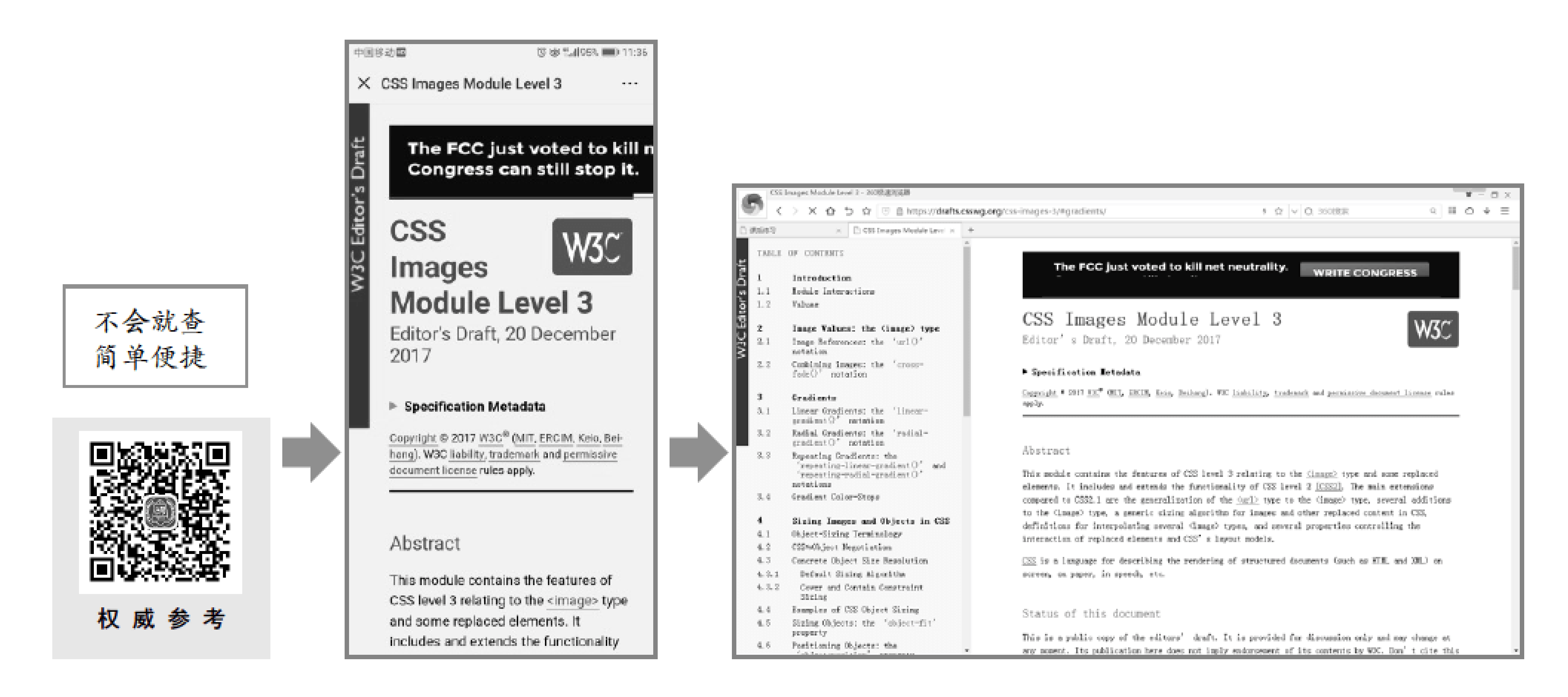






6. 查阅权威参考资料

扫描“权威参考”二维码，即可跳转到对应知识的官方文档上。通过大量查阅，真正领悟技术内涵。



7. 其他 PC 端资源下载方式

除了前面介绍过的可以直接将视频、拓展阅读等资源推送到邮箱之外，还提供了如下几种 PC 端资源获取方式。

☑ 登录清华大学出版社官方网站（www.tup.com.cn），在对应图书页面下查找资源的下载方式。

☑ 申请加入 QQ 群、微信群，获得资源的下载方式。

☑ 扫描图书封底二维码（文泉云盘），获得资源的下载方式。

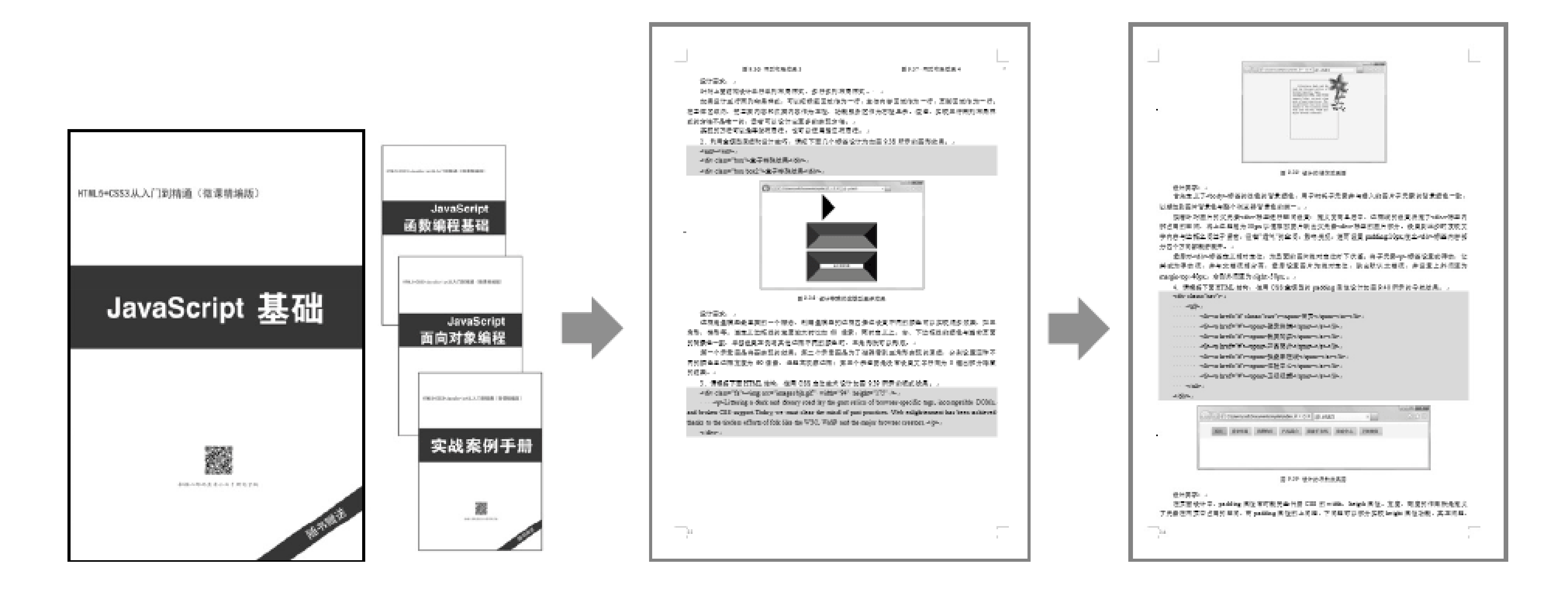
小白实战手册

为方便读者全面提升，本书提供了 4 本小白学习手册：JavaScript 基础、JavaScript 函数编程基础、JavaScript 面向对象编程以及实战案例手册。这些内容精挑细选，希望成为您学习路上的好帮手，关键时刻解您所需。



扫描小白手册封面的二维码，可在手机、平板上学习小白手册内容。





从小白到高手的蜕变

谷歌的创始人拉里•佩奇说过，如果你刻意练习某件事超过 10000 个小时，那么你就可以达到世界级。

因此，不管您现在是怎样的前端开发小白，只要您按着下面的步骤来学习，假以时日，您会成为令自己惊讶的技术大咖。

（1）扎实的基础知识+大量的中小实例训练+有针对性地做一些综合案例。

（2）大量的项目案例观摩、学习、操练，塑造一定的项目思维。

（3）善于借用他山之石，对一些成熟的开源代码、设计素材拿来就用，学会站在巨人的肩膀上。

（4）有功夫多参阅一些官方权威指南，拓展自己对技术的理解和应用能力。

（5）最为重要的是，多与同行交流，在切磋中不断进步。

书本厚度有限，学习空间无限。纸张价格有限，知识价值无限。希望本书能帮您真正收获学习的乐趣和知识。最后，祝您阅读快乐！

前　言

Preface

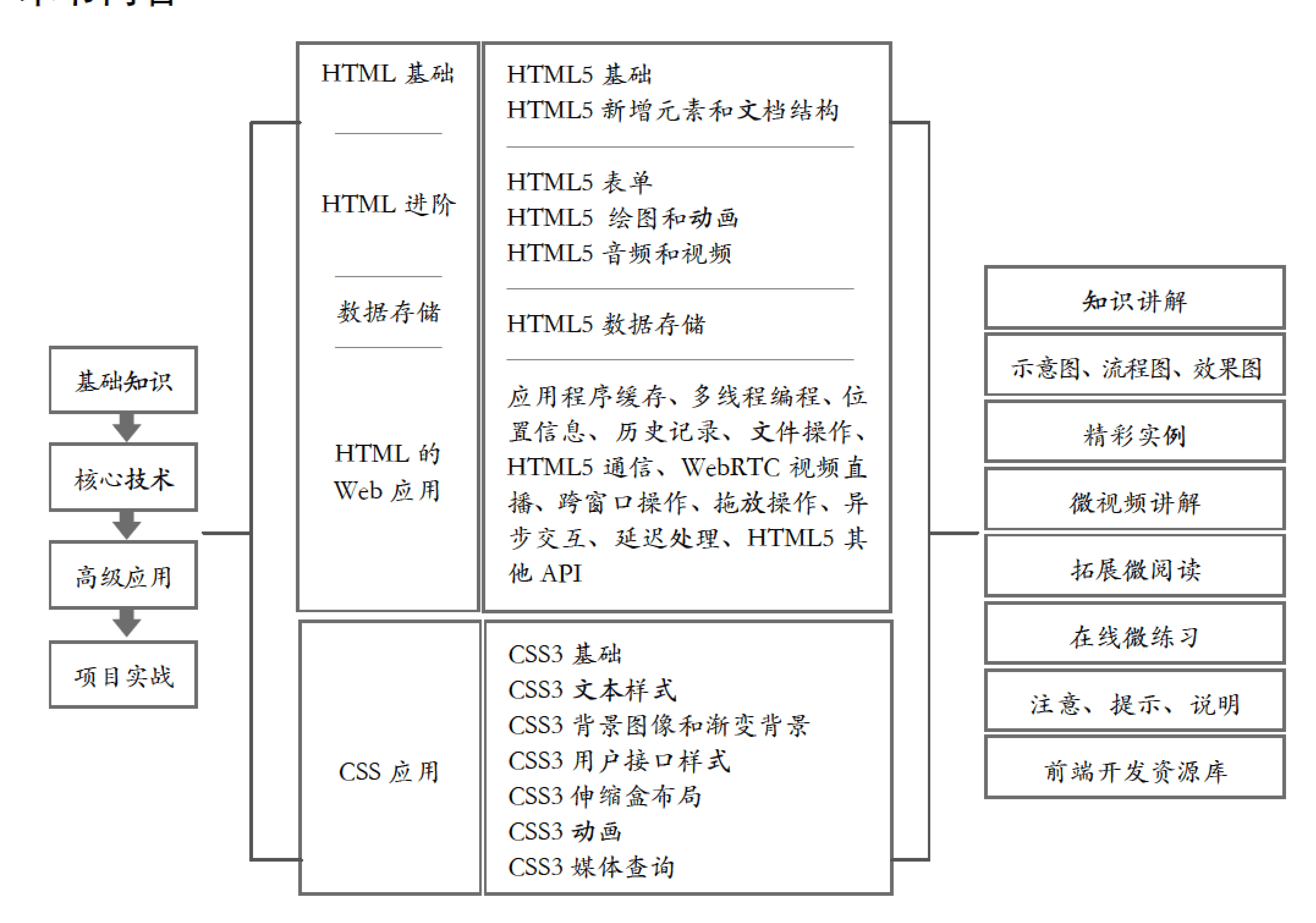
“网络开发视频大讲堂”系列丛书于 2013 年 5 月出版，因其编写细腻、讲解透彻、实用易学、配备全程视频等，备受读者欢迎。丛书累计销售近 20 万册，其中，《HTML5+CSS3 从入门到精通》累计销售 10 万册。同时，系列书被上百所高校选为教学参考用书。

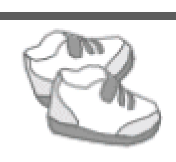
本次改版，在继承前版优点的基础上，进一步对图书内容进行了优化，选择面试、就业最急需的内容，重新录制了视频，同时增加了许多当前流行的前端技术，提供了“入门学习→实例应用→项目开发→能力测试→面试”等各个阶段的海量开发资源库，实战容量更大，以帮助读者快速掌握前端开发所需要的核心精髓内容。

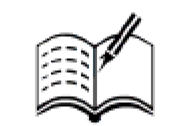
网页制作技术可以粗略划分为前台浏览器端技术和后台服务器端技术。在Web标准中，HTML负责页面结构，CSS负责样式表现。

网页技术层出不穷，日新月异，但有一点是肯定的：不管是采用什么技术设计的网站，用户在客户端通过浏览器打开看到的网页都是静态网页，都是由HTML和CSS技术构成的。因此，HTML和CSS技术是网页制作技术的基础和核心。

本书内容







本书特点

1. 由浅入深，编排合理，实用易学

本书面向零基础的初学者，通过“一个知识点+一个例子+一个结果+一段评析+一个综合应用”的写作模式，全面、细致地讲述了HTML5+CSS3实际开发中所需的各类知识，由浅入深，循序渐进。同时，本书展示了许多Web时代备受欢迎的新知识，读者可学习到与HTML5相关的一些非常实用、流行的技术。

2. 跟着案例和视频学，入门更容易

跟着例子学习，通过训练提升，是初学者最好的学习方式。本书案例丰富详尽，多达550个，且都附有详尽的代码注释及清晰的视频讲解。跟着这些案例边做边学，可以避免学到的知识流于表面、限于理论，尽情感受编程带来的快乐和成就感。

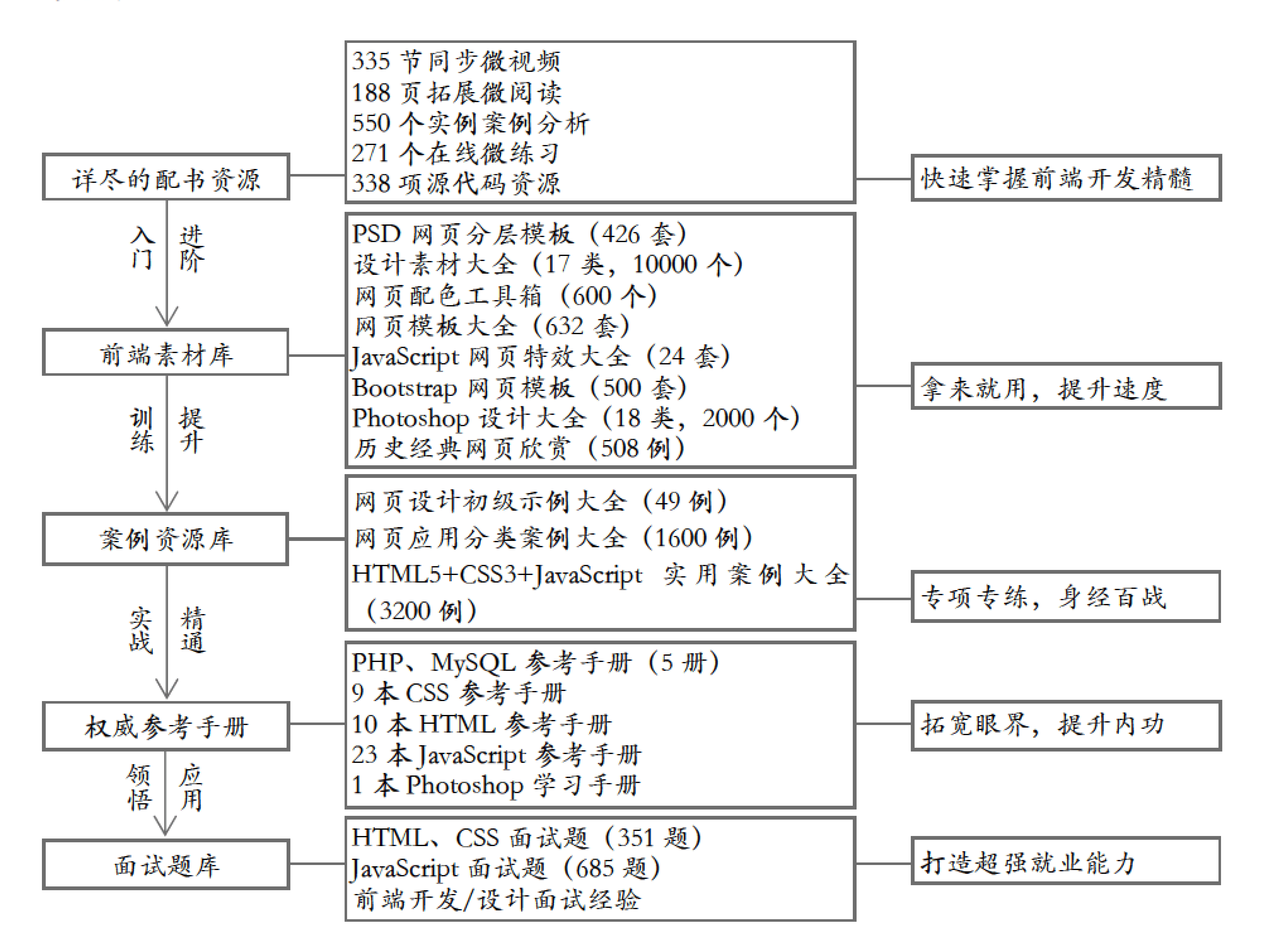
3. 5大类线上资源，多元化学习体验

为了传递更多知识，本书力求突破传统纸质书的厚度限制。本书提供了5大类线上微资源，通过手机扫码，读者可随时观看讲解视频，拓展阅读相关知识，在线练习强化提升，还可以欣赏动态案例效果和查阅官方权威资料，全程便捷、高效，感受不一样的学习体验。

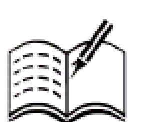
4. 精彩栏目，易错点、重点、难点贴心提醒

本书根据初学者特点，在一些易错点、重点、难点位置精心设置了“注意”“提示”等小栏目。通过这些小栏目，读者会更留心相关的知识点和概念，绕过陷阱，掌握很多应用技巧。

本书资源







读者对象

- ☑ 零基础的编程自学者。
- ☑ 相关培训机构的老师和学生。
- ☑ 大中专院校的老师和学生。
- ☑ 参加毕业设计的学生。
- ☑ 初、中级程序开发人员。

读前须知

作为入门书籍，本书知识点比较庞杂，所以不可能面面俱到。技术学习的关键是方法，本书在很多实例中体现了方法的重要性，读者只要掌握了各种技术的运用方法，在学习更深入的知识中可大大提高自学的效率。

本书提供了大量示例，需要用到IE、Firefox、Chrome、Opera等主流浏览器的测试和预览。因此，为了测试示例或代码，读者需要安装上述类型的最新版本浏览器，各种浏览器在CSS3的表现上可能会稍有差异。

限于篇幅，本书示例没有提供完整的HTML代码，读者应该补充完整的HTML结构，然后进行测试练习，或者直接参考本书提供的下载源代码，边学边练。

为了给读者提供更多的学习资源，本书提供了很多参考链接，许多本书无法详细介绍的问题都可以通过这些链接找到答案。由于这些链接地址会因时间而有所变动或调整，所以在此说明，这些链接地址仅供参考，本书无法保证所有的这些地址是长期有效的。

读者服务

学习本书时，请先扫描封底的权限二维码（需要刮开涂层）获取学习权限，然后即可免费学习书中的所有线上线下资源。

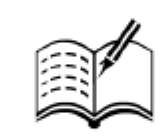
本书所附赠的超值资源库内容，读者可登录清华大学出版社网站（www.tup.com.cn），在对应图书页面下获取其下载方式。也可扫描图书封底的“文泉云盘”二维码，获取其下载方式。

本书提供QQ群（668118468、697651657）、微信群（qianduankaifa_cn）、服务网站（www.qianduankaifa.cn）等互动渠道，提供在线技术交流、学习答疑、技术资讯、视频课堂、在线勘误等功能。在这里，您可以结识大量志同道合的朋友，在交流和切磋中不断成长。

读者对本书有什么好的意见和建议，也可以通过邮箱（qianduanjiaoshi@163.com）发邮件给我们。

关于作者

前端科技是由一群热爱Web开发的青年骨干教师和一线资深开发人员组成的一个团队，主要从事Web开发、教学和培训。参与本书编写的人员包括咸建勋、奚晶、文菁、李静、钟世礼、袁江、甘桂萍、刘燕、杨凡、朱砚、余乐、邹仲、余洪平、谭贞军、谢党华、何子夜、赵美青、牛金鑫、孙玉静、左超红、蒋学军、邓才兵、陈文广、李东博、林友赛、苏震巍、崔鹏飞、李斌、郑伟、邓艳超、胡晓



Note

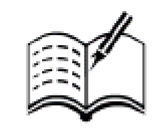
霞、朱印宏、刘望、杨艳、顾克明、郭靖、朱育贵、刘金、吴云、赵德志、张卫其、李德光、刘坤、彭方强、雷海兰、王鑫铭、马林、班琦、蔡霞英、曾德剑等。

尽管已竭尽全力，但由于水平有限，书中疏漏和不足之处在所难免，欢迎各位读者朋友批评、指正。

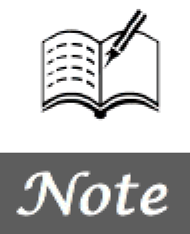
编者

2018 年 6 月

目　录

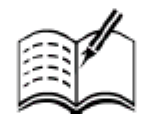


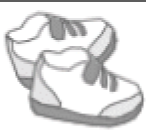
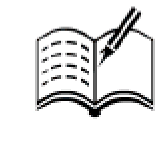
Contents



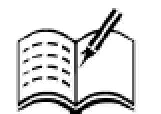
Note








Note



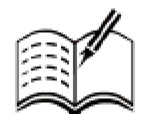


Note




Note

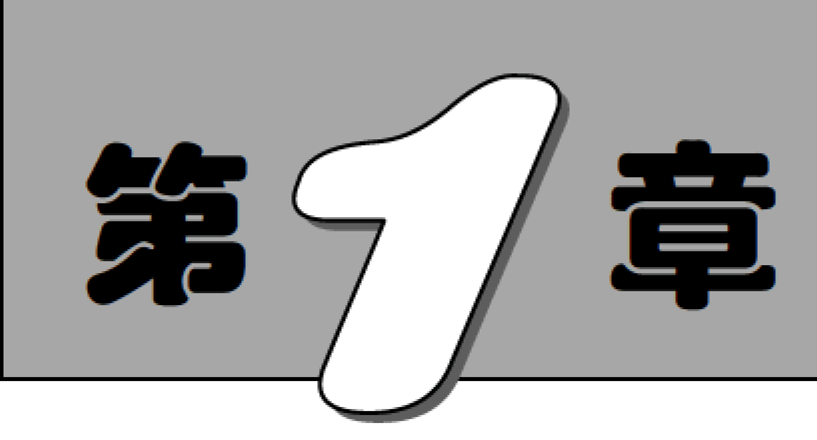



Note





Note



第1章 HTML5基础

2014年10月28日，W3C的HTML工作组发布了HTML5的正式推荐标准。HTML5是构建开放Web平台的核心，是万维网的核心语言——可扩展标记语言的第5版。在这一版本中，增加了支持Web应用的许多新特性，以及更符合开发者使用习惯的新元素，并重点关注定义清晰的、一致的准则，以确保Web应用和内容在不同浏览器中的互操作性。

本章将对HTML5进行简单概述，对于继承自HTML4的大部分内容就不再赘述，有关HTML5 API部分将在后面各章中逐步展开讲解。

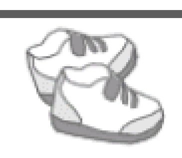
权威参考：https://www.w3.org/TR/html5/

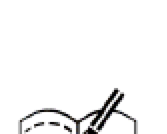


权威参考

【学习重点】

- 了解HTML版本和HTML5开发历史。
- 了解HTML5设计理念。
- 了解HTML5 API。
- 熟悉HTML5基本语法。





1.1 HTML5 概述

从 2010 年开始，HTML5 和 CSS3 就一直是网络世界倍受追捧的技术热点。以 HTML5+CSS3 为主的网络时代，使互联网进入了一个崭新的发展阶段。

1.1.1 HTML 版本概览

HTML 从诞生至今，经历了近 30 年的发展，其中经历的版本及发布日期如表 1.1 所示。

表 1.1 HTML 语言的发展过程

版　本	发布日期	说　明
超文本标记语言（第一版）	1993 年 6 月	作为互联网工程工作小组（IETF）工作草案发布，非标准
HTML 2.0	1995 年 11 月	作为 RFC 1866 发布，在 RFC 2854 于 2000 年 6 月发布之后被宣布已经过时
HTML 3.2	1996 年 1 月 14 日	W3C 推荐标准
HTML 4.0	1997 年 12 月 18 日	W3C 推荐标准
HTML 4.01	1999 年 12 月 24 日	微小改进，W3C 推荐标准
ISO HTML	2000 年 5 月 15 日	基于严格的 HTML4.01 语法，是国际标准化组织和国际电工委员会的标准
XHTML 1.0	2000 年 1 月 26 日	W3C 推荐标准，修订后于 2002 年 8 月 1 日重新发布
XHTML 1.1	2001 年 5 月 31 日	较 1.0 有微小改进
XHTML 2.0 草案	没有发布	2009 年，W3C 停止了 XHTML 2.0 工作组的工作
HTML5 草案	2008 年 1 月	HTML5 规范先是以草案发布，经历了漫长的过程
HTML5.1	2017 年 10 月 3 日	W3C 发布 HTML5 第 1 个更新版本（http://www.w3.org/TR/html51/）
HTML5.2	2017 年 12 月 14 日	W3C 发布 HTML5 第 2 个更新版本（http://www.w3.org/TR/html52/）
HTML5.3	2018 年 3 月 15 日	W3C 发布 HTML5 第 3 个更新版本（http://www.w3.org/TR/html53/）

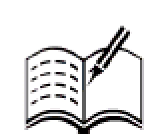
提示： 从上面 HTML 发展列表来看，HTML 没有 1.0 版本，这主要是因为当时有很多不同的版本。有些人认为 Tim Berners-Lee 的版本应该算初版，其版本中还没有 img 元素，也就是说，HTML 刚开始仅能够显示文本信息。

1.1.2 HTML5 诞生记

在 20 世纪末期，W3C 开始琢磨着改良 HTML 语言，当时的版本是 HTML4.01。但是在后来的开发和维护过程中，出现了方向性分歧：是开发 XHTML 1，再到 XHTML 2，最后终极目标是 XML；还是坚持实用主义原则，快速开发出改良的 HTML5 版本？

2004 年 W3C 成员内部的一次研讨会上，当时 Opera 公司的代表伊恩・希克森（Ian Hickson）提出了一个扩展和改进 HTML 的建议。他建议新任务组可以跟 XHTML 2 并行，但是在已有 HTML 的基础上开展工作，目标是对 HTML 进行扩展。但是 W3C 投票表示反对，因为他们认为 HTML 已经





毫无前景，XHTML 2 才是未来的方向。

然后，Opera、Apple 等浏览器厂商，以及部分成员忍受不了 W3C 的工作机制和拖沓的行事节奏，决定脱离 W3C，他们成立了 WHATWG （Web Hypertext Applications Technology Working Group，Web 超文本应用技术工作组），这就为 HTML5 将来的命运埋下了伏笔。

WHATWG 决定完全脱离 W3C，在 HTML 的基础上开展工作，向其中添加一些新东西。这个工作组的成员里有浏览器厂商，因此他们可以保证实现各种新奇、实用的点子。结果，大家不断提出一些好点子，并且逐一整合到新版本浏览器中。

WHATWG 的工作效率很高，不久就初见成效。在此期间，W3C 的 XHTML 2 没有什么实质性的进展。在 2006 年，蒂姆 • 伯纳斯-李写了一篇博客反思 HTML 发展历史："你们知道吗？我们错了。我们错在企图一夜之间就让 Web 跨入 XML 时代，我们的想法太不切实际了，是的，也许我们应该重新组建 HTML 工作组了。"

W3C 在 2007 年组建了 HTML5 工作组。这个工作组面临的第一个问题是"我们是从头开始做起呢，还是在 2004 年成立的那个叫 WHATWG 的工作组既有成果的基础上开始工作呢？"

答案是显而易见的，他们当然希望从已经取得的成果着手，以此为基础开展工作。工作组投了一次票，同意在 WHATWG 工作成果的基础上继续开展工作。

第二个问题就是如何理顺两个工作组之间的关系。W3C 这个工作组的编辑应该由谁担任？是不是还让 WHATWG 的编辑，也就是现在 Google 的伊恩 • 希克森来兼任？于是他们又投了一次票，赞成让伊恩 • 希克森担任 W3C HTML5 规范的编辑，同时兼任 WHATWG 的编辑，更有助于新工作组开展工作。

这就是他们投票的结果，也就是我们今天看到的局面：一种格式，两个版本。WHATWG 网站上有这个规范，而 W3C 网站上同样也有一份。

如果不了解内情，你很可能会产生这样的疑问："哪个版本才是真正的规范？"当然，这两个版本内容是一样的，基本上相同。实际上，这两个版本将来还会分道扬镳。现在已经有了分道扬镳的迹象。W3C 最终要制定一个具体的规范，这个规范会成为一个工作草案，定格在某个历史时刻。

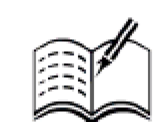
而 WHATWG 还在不断地迭代。即使目前的 HTML5 也不能完全涵盖 WHATWG 正在从事的工作。最准确的理解就是 WHATWG 正在开发一项简单的 HTML 或 Web 技术，因为这才是他们工作的核心目标。然而，同时存在两个这样的工作组，这两个工作组同时开发一个基本相同的规范，这无论如何也容易让人产生误解，误解就可能造成麻烦。

其实这两个工作组背后各自有各自的流程，因为它们的理念完全不同。在 WHATWG 内部，可以说是一种独裁的工作机制。伊恩 • 希克森是编辑。他会听取各方意见，在所有成员各抒己见，充分陈述自己的观点之后，他批准自己认为正确的意见。而 W3C 则截然相反，可以说是一种民主的工作机制。所有成员都可以发表意见，而且每个人都有投票表决的权利。这个流程的关键在于投票表决。从表面上看，WHATWG 的工作机制让人难以接受，W3C 的工作机制听起来让人很舒服，至少体现了人人平等的精神。但在实践中，WHATWG 的工作机制运行得非常好。这主要归功于伊恩 • 希克森。他在听取各方意见时，始终可以做到丝毫不带个人感情色彩。

从原理上讲，W3C 的工作机制很公平，而实际上却非常容易在某些流程或环节上卡壳，造成工作停滞不前，一件事情要达成决议往往需要花费很长时间。那到底哪种工作机制最好呢？个人认为，最好的工作机制是将二者结合起来。而事实也是两个规范制定主体在共同制定一份相同的规范，这倒是非常有利于两种工作机制相互取长补短。

两个工作组之所以能够同心同德，主要原因是 HTML5 的设计思想。因为从一开始就确定了设计 HTML5 所要坚持的原则。结果，我们不仅看到了一份规范，也就是 W3C 站点上公布的那份文档，即 HTML5 语言规范，还在 W3C 站点上看到了另一份文档，也就是 HTML5 设计原理。





Note

1.1.3 HTML5 组织

HTML5 是 W3C 与 WHATWG 合作的结晶。HTML5 开发主要由下面三个组织负责。

☑ WHATWG：由来自 Apple、Mozilla、Google、Opera 等浏览器厂商的专家组成，成立于 2004 年。WHATWG 负责开发 HTML 和 Web 应用 API。

☑ W3C：指 World Wide Web Consortium，万维网联盟，负责发布 HTML5 规范。

☑ IETF（因特网工程任务组）：负责 Internet 协议开发。HTML5 定义的 WebSocket API 依赖于新的 WebSocket 协议，IETF 工作组负责开发这个协议。

1.1.4 HTML5 开发规则

为了避免 HTML5 开发过程中出现的各种分歧和偏差，HTML5 开发工作组在共识基础上建立了一套行事规则：

☑ 新特性应该基于 HTML、CSS、DOM 以及 JavaScript。

☑ 减少对外部插件的依赖，如 Flash。

☑ 更优秀的错误处理。

☑ 更多取代脚本的标记。

☑ HTML5 应该独立于设备。

☑ 开发进程应即时、透明，倾听技术社区的声音，吸纳社区内优秀的 Web 应用。

☑ 允许试错，允许纠偏，从实践中来，服务于实践，快速迭代。

1.1.5 HTML5 特性

下面简单介绍 HTML5 特性和优势，以便增强读者自学 HTML5 的动力和明确目标。

1．兼容性

考虑到互联网上 HTML 文档已经存在 20 多年了，因此支持所有现存 HTML 文档是非常重要的。HTML5 不是颠覆性的革新，它的核心理念就是要保持与过去技术的兼容和过渡。一旦浏览器不支持 HTML5 的某项功能，针对该功能的备选行为就会悄悄运行。

2．实用性

HTML5 新增加的元素都是对现有网页和用户习惯进行跟踪、分析和概括而推出的。例如，Google 分析了上百万的页面，从中分析出了 DIV 标签的通用 ID 名称，并且发现其重复量很大，如很多开发人员使用<div id="header">来标记页眉区域，为了解决实际问题，HTML5 就直接添加一个<header>标签。也就是说，HTML5 新增的很多元素、属性或者功能都是根据现实互联网中已经存在的各种应用进行归纳和提炼，而不是在实验室中进行理想化的虚构新功能。

3．效率

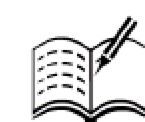
HTML5 规范是基于用户优先的原则编写的，其宗旨是用户即上帝，这意味着在遇到无法解决的冲突时，规范会把用户放到第一位，其次是页面制作者，再次是浏览器解析标准，接着是规范制定者（如 W3C、WHATWG），最后才考虑理论的纯粹性。因此，HTML5 的绝大部分是实用的，只是有些情况下还不够完美。例如，下面的几种代码写法在 HTML5 中都能被识别。

```
id="prohtml5"
```



```
id=prohtml5
ID="prohtml5"
```

当然，上面几种写法比较混乱，不够严谨，但是从用户开发角度考虑，用户不在乎代码怎么写，根据个人书写习惯反而提高了代码编写效率。



4．安全性

为保证足够安全，HTML5 引入了一种新的基于来源的安全模型，该模型不仅易用，而且对各种不同的 API 都通用。这个安全模型可以不需要借助于任何所谓聪明、有创意却不安全的 hack 就能跨域进行安全对话。

5．分离

在清晰分离表现与内容方面，HTML5 迈出了很大的步伐。HTML5 在所有可能的地方都努力进行了分离，包括 HTML 和 CSS。实际上，HTML5 规范已经不支持老版本 HTML 的大部分表现功能了。

6．简化

HTML5 要的就是简单、避免不必要的复杂性。HTML5 的口号是：简单至上，尽可能简化。因此，HTML5 做了以下改进：

- ☑ 以浏览器原生能力替代复杂的 JavaScript 代码。
- ☑ 简化的 DOCTYPE。
- ☑ 简化的字符集声明。
- ☑ 简单而强大的 HTML5 API。

7．通用性

通用访问的原则可以分成三个概念：

- ☑ 可访问性：出于对残障用户的考虑，HTML5 与 WAI（Web 可访问性倡议）和 ARIA（可访问的富 Internet 应用）做到了紧密结合，WAI-ARIA 中以屏幕阅读器为基础的元素已经被添加到 HTML 中。
- ☑ 媒体中立：如果可能的话，HTML5 的功能在所有不同的设备和平台上应该都能正常运行。
- ☑ 支持所有语种：如新的<ruby>元素支持在东亚页面排版中会用到的 Ruby 注释。

8．无插件

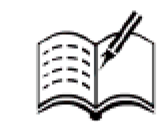
在传统 Web 应用中，很多功能只能通过插件或者复杂的 hack 来实现，但在 HTML5 中提供了对这些功能的原生支持。插件的方式存在很多问题：

- ☑ 插件安装可能失败。
- ☑ 插件可以被禁用或屏蔽，如 Flash 插件。
- ☑ 插件自身会成为被攻击的对象。
- ☑ 因为插件边界、剪裁和透明度问题，插件不容易与 HTML 文档的其他部分集成。

以 HTML5 中的 canvas 元素为例，有很多非常底层的事情以前是没办法做到的，如在 HTML4 的页面中就难画出对角线，而有了 canvas 就可以很轻易地实现了。基于 HTML5 的各类 API 的优秀设计，可以轻松地对它们进行组合应用。例如，从 video 元素中抓取的帧可以显示在 canvas 里面，用户单击 canvas 即可播放这帧对应的视频文件。

最后，用万维网联盟创始人蒂姆·伯纳斯-李的评论来小结，“今天，我们想做的事情已经不再是通过浏览器观看视频或收听音频，或者在一部手机上运行浏览器。我们希望通过不同的设备，在任何





地方，都能够共享照片、网上购物、阅读新闻以及查找信息。虽然大多数用户对 HTML5 和开放 Web 平台（Open Web Platform，OWP）并不熟悉，但是它们正在不断改进用户体验”。

1.1.6 浏览器检测

HTML5 发展的速度非常快，因此不用担心浏览器的支持问题。用户可以访问 www.caniuse.com，该网站按照浏览器的版本提供了详尽的 HTML5 功能支持情况。

如果通过浏览器访问 www.html5test.com，该网站会直接显示用户浏览器对 HTML5 规范的支持情况。另外，还可以使用 Modernizr（JavaScript 库）进行特性检测，它提供了非常先进的 HTML5 和 CSS3 检测功能。建议使用 Modernizr 检测当前浏览器是否支持某些特性。

1.2 HTML5 设计原则

从 HTML 2.0 到 XHTML 2.0，XHTML 2.0 由于语法解析过于严格，因此不太适合互联网开放、自由的精神。Jeremy Keith 认为所有的项目都应该先有设计原则，HTML5 也同样如此，W3C 为此发布了 HTML 设计原则（http://www.w3.org/TR/html-design-principles/），强调了 HTML5 规范的兼容性、实用性和互操作性。

1.2.1 避免不必要的复杂性

规范可以写得十分复杂，但浏览器的实现应该非常简单。把复杂的工作留给浏览器后台去处理，用户仅需要输入简单的字符，甚至不需要输入，才是最佳文档规范。因此，HTML5 首先采用化繁为简的思路进行设计。

【示例 1】在 HTML4.01 中定义文档类型的代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C/DTD HTML4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

HTML5 简化如下：

```
<!DOCTYPE html>
```

HTML4.01 和 XHTML 中的 DOCTYPE 过于冗长，连自己都记不住这些内容，但在 HTML5 中只需要简单的<!DOCTYPE html>就可以了。DOCTYPE 是给验证器用的，而非浏览器，浏览器只在做 DOCTYPE 切换时关注这个标签，因此并不需要写得太复杂。

【示例 2】在 HTML4.01 中定义字符编码的代码如下：

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

在 XHTML 1.0 中还需要再声明 XML 标签，并在其中指定字符编码。

```
<?xml version="1.0" encoding="UTF-8" ?>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

HTML5 简化如下：

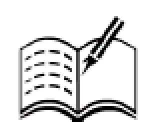
```
<meta charset="utf-8">
```

关于省略不必要的复杂性，或者说避免不必要的复杂性的例子还有不少，但关键是既能避免不必



要的复杂性，还不会妨碍在现有浏览器中使用。

在 HTML5 中，如果使用 link 元素链接到一个样式表，先定义 rel="stylesheet"，然后再定义 type="text/css"，这样就重复了。对浏览器而言，只要设置 rel="stylesheet"就够了，因为它可以猜出要链接的是一个 CSS 样式表，没必要再指定 type 属性。



对 Web 开发而言，大家都使用 JavaScript 脚本语言，也是默认的通用语言，用户可以为 script 元素定义 type="text/javascript"属性，也可以什么都不写，浏览器自然会假设在使用 JavaScript。

1.2.2　支持已有内容

XHTML 2.0 最大的问题就是不支持已经存在的内容，这违反了 Postel 法则（即对自己发送的东西要严格，对接收的东西则要宽容）。现实情况中，开发者可以写出各种风格的 HTML，浏览器遇到这些代码时，在内部所构建出的结构应该是一样的，呈现的效果也应该是一样的。

【示例】下面示例展示了编写同样内容的四种不同写法，四种写法唯一的不同点就是语法。

```
<img src="foo" alt="bar" />
<p class="foo">Hello world</p>

<img src="foo" alt="bar">
<p class="foo">Hello world

<IMG SRC="foo" ALT="bar">
<P CLASS="foo">Hello world</P>

<img src=foo alt=bar>
<p class=foo>Hello world</p>
```

从浏览器解析的角度分析，这些写法实际上都是一样的。HTML5 必须支持已经存在的约定，适应不同的用户习惯，而不是用户适应浏览器的严格解析标准。

1.2.3　解决实际问题

规范应该去解决现实中实际遇到的问题，而不该考虑那些复杂的理论问题。

【示例】既然有在<a>中嵌套多个段落标签的需要，那就让规范支持它。

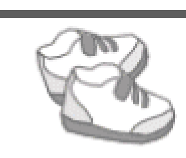
这块内容包含一个标题和一个段落。按 HTML4 规范，必须至少使用两个链接。

```
<h2><a href="#">标题文本</a></h2>
<p><a href="#">段落文本</a></p>
```

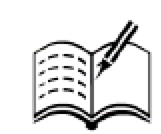
在 HTML5 中，只需要把所有内容都包裹在一个链接中就行了。

```
<a href="#">
    <h2>标题文本</h2>
    <p>段落文本</p>
</a>
```

其实这种写法早已经存在，当然以前这样写是不合乎规范的。所以说，HTML5 解决现实的问题，其本质还是纠正因循守旧的规范标准，现在把标准改了，允许用户这样写了。



1.2.4 用户怎么使用就怎么设计规范


Note

当一个实践已经被广泛接受时，就应该考虑将它吸纳进来，而不是禁止它或开发一个新的实践。例如，HTML5 新增了 nav、section、article、aside 等标签，它们引入了新的文档模型，即文档中的文档。在 section 中，还可以嵌套 h1 到 h6 的标签，这样就有了无限的标题层级，这也是很早之前 Tim Berners Lee 所设想的。

【示例】下面几行代码相信大家都不会陌生，这些都是频繁被使用过的 ID 名称。

```
<div id="header">...</div>
<div id="navigation">...</div>
<div id="main">...</div>
<div id="aside">...</div>
<div id="footer">...</div>
```

在 HTML5 中，可以用新的元素代替使用。

```
<header>...</header>
<nav>...</nav>
<div id="main">...</div>
<aside>...</aside>
<footer>...</footer>
```

实际上，这并不是 HTML5 工作组想出来的，也不是 W3C 提出来的，而是谷歌根据大数据分析用户习惯得出来的。

1.2.5 优雅地降级

渐进增强的另一面就是优雅地回退。最典型的例子就是使用 type 属性增强表单。

【示例 1】下面代码列出了可以为 type 属性指定的新值，如 number、search、range 等。

```
<input type="number" />
<input type="search" />
<input type="range" />
<input type="email" />
<input type="date" />
<input type="url" />
```

最关键的问题在于：当浏览器看到这些新 type 值时会如何处理。老版本浏览器是无法理解这些新 type 值的。但是当它们看到自己不理解的 type 值时，会将 type 的值解释为 text。

【示例 2】对于新的 video 元素，它设计得很简单、实用。针对不支持 video 元素的浏览器可以这样写。

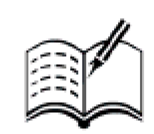
```
<video src="movie.mp4">
    <!-- 回退内容 -->
</video>
```

这样 HTML5 视频与 Flash 视频就可以协同起来，用户不用纠结如何选择。

```
<video src="movie.mp4">
    <object data="movie.swf">
```



```
        <!-- 回退内容 -->
    </object>
</video>
```



如果愿意的话，还可以使用 source 元素，而非 src 属性来指定不同的视频格式。

```
<video>
    <source src="movie.mp4">
    <source src="movie.ogv">
    <object data="movie.swf">
        <a href="movie.mp4">download</a>
    </object>
</video>
```

上面代码包含了 4 个不同的层次。

- ☑ 如果浏览器支持 video 元素，也支持 H264，没什么好说的，用第一个视频。
- ☑ 如果浏览器支持 video 元素，支持 Ogg，那么用第二个视频。
- ☑ 如果浏览器不支持 video 元素，那么就要试试 Flash 视频。
- ☑ 如果浏览器不支持 video 元素，也不支持 Flash 视频，还可以给出下载链接。

总之，无论是 HTML5，还是 Flash，一个也不能少。如果只使用 video 元素提供视频，难免会遇到问题。而如果只提供 Flash 影片，性质是一样的。所以还是应该两者兼顾。

1.2.6　支持的优先级

用户与开发者的重要性要远远高于规范和理论。在考虑优先级时，应该按照下面顺序：

用户 > 编写 HTML 的开发者 > 浏览器厂商 > 规范制定者 > 理论

这个设计原则本质上是一种解决冲突的机制。例如，当面临一个要解决的问题时，如果 W3C 给出了一种解决方案，而 WHATWG 给出了另一种解决方案。一旦遇到冲突，最终用户优先，其次是作者，再是实现者，然后标准制定者，最后才是理论上的完美。

根据最终用户优先的原理，开发人员在链条中的位置高于实现者，假如我们发现了规范中的某些地方有问题，就不支持实现这个特性，那么就等于把相应的特性给否定了，规范里就得删除，因为用户有更高的权重。本质上用户拥有了更大的发言权，开发人员也拥有更多的主动性。

1.3　HTML5 语法特性

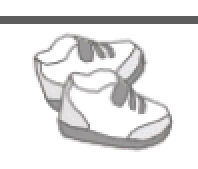
HTML5 以 HTML4 为基础，对 HTML4 进行了全面升级改造。与 HTML4 相比，HTML5 在语法上有很大的变化，具体比较如下：

1.3.1　文档和标记

1．内容类型

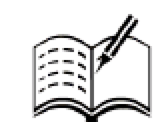
HTML5 的文件扩展名和内容类型保持不变。例如，扩展名仍然为“.html”或“.htm”，内容类型（ContentType）仍然为“text/html”。





2．文档类型

在 HTML4 中，文档类型的声明方法如下：



Note

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

在 HTML5 中，文档类型的声明方法如下：

```
<!DOCTYPE html>
```

当使用工具时，也可以在 DOCTYPE 声明中加入 SYSTEM 识别符，声明方法如下：

```
<!DOCTYPE HTML SYSTEM "about:legacy-compat">
```

在 HTML5 中，DOCTYPE 声明方式是不区分大小写的，引号也不区分是单引号还是双引号。

> 注意：使用 HTML5 的 DOCTYPE 会触发浏览器以标准模式显示页面。众所周知，网页都有多种显示模式，如怪异模式（Quirks）、标准模式（Standards）。浏览器根据 DOCTYPE 来识别该使用哪种解析模式。

3．字符编码

在 HTML4 中，使用 meta 元素定义文档的字符编码，如下所示：

```
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
```

在 HTML5 中，继续沿用 meta 元素定义文档的字符编码，但是简化了 charset 属性的写法，如下所示：

```
<meta charset="UTF-8">
```

对于 HTML5 来说，上述两种方法都有效，用户可以继续使用前面一种方式，即通过 content 元素的属性来指定。但是不能同时混用两种方式。

> 注意：在传统网站中，可能会存在下面的标记。在 HTML5 中，这种字符编码方式将被认为是错误的。
> ```
> <meta charset="UTF-8" http-equiv="Content-Type" content="text/html;charset=UTF-8">
> ```

从 HTML5 开始，对于文件的字符编码推荐使用 UTF-8。



视 频 讲 解

1.3.2　宽松的约定

HTML5 语法是为了保证与之前的 HTML4 语法达到最大限度的兼容而设计的。

1．标记省略

在 HTML5 中，元素的标记可以分为三种类型：不允许写结束标记，可以省略结束标记，开始标记和结束标记全部可以省略。下面简单介绍这三种类型各包括哪些 HTML5 新元素。

第一，不允许写结束标记的元素有：area、base、br、col、command、embed、hr、img、input、keygen、link、meta、param、source、track、wbr。

第二，可以省略结束标记的元素有：li、dt、dd、p、rt、rp、optgroup、option、colgroup、thead、tbody、tfoot、tr、td、th。



第三，可以省略全部标记的元素有：html、head、body、colgroup、tbody。



提示：不允许写结束标记的元素是指，不允许使用开始标记与结束标记将元素括起来的形式，只允许使用<元素/>的形式进行书写。例如：

☑　错误的书写方式

</br>

☑　正确的书写方式



HTML5 之前的版本中
这种写法可以继续沿用。

可以省略全部标记的元素是指元素可以完全被省略。注意，该元素还是以隐式的方式存在的。例如，将 body 元素省略时，它在文档结构中还是存在的，可以使用 document.body 进行访问。

2．布尔值

对于布尔型属性，如 disabled 与 readonly 等，当只写属性而不指定属性值时，表示属性值为 true；如果属性值为 false，可以不使用该属性。另外，要想将属性值设定为 true，也可以将属性名设定为属性值，或将空字符串设定为属性值。

【示例 1】下面是几种正确的书写方法：

```
<!--只写属性，不写属性值，代表属性为 true-->
<input type="checkbox" checked>
<!--不写属性，代表属性为 false-->
<input type="checkbox">
<!--属性值=属性名，代表属性为 true-->
<input type="checkbox" checked="checked">
<!--属性值=空字符串，代表属性为 true-->
<input type="checkbox" checked="">
```

3．属性值

属性值可以加双引号，也可以加单引号。HTML5 在此基础上做了一些改进，当属性值不包括空字符串、<、>、=、单引号、双引号等字符时，属性值两边的引号可以省略。

【示例 2】下面写法都是合法的：

```
<input type="text">
<input type='text'>
<input type=text>
```

1.4　案例实战

目前主流浏览器对 HTML5 提供了很好的支持，下面结合示例介绍如何正确创建 HTML5 文档。

1.4.1　编写第一个 HTML5 文档



本节示例将遵循 HTML5 语法规范编写一个文档。本例文档省略了<html>、<head>、<body>等标





Note

签，使用 HTML5 的 DOCTYPE 声明文档类型，简化<meta>的 charset 属性设置，省略<p>标签的结束标记、使用<元素/>的方式来结束<meta>和
标签等。

```
<!DOCTYPE html>
<meta charset="UTF-8">
<title>HTML5 基本语法</title>
<h1>HTML5 的目标</h1>
<p>HTML5 的目标是为了能够创建更简单的 Web 程序，书写出更简洁的 HTML 代码。
<br/>例如，为了使 Web 应用程序的开发变得更容易，提供了很多 API；为了使 HTML 变得更简洁，开发出了新的属性、新的元素等。总体来说，为下一代 Web 平台提供了许许多多新的功能。
```

这段代码在 IE 浏览器中的运行结果如图 1.1 所示。





示例效果

图 1.1　编写 HTML5 文档

通过短短几行代码就完成了一个页面的设计，这充分说明了 HTML5 语法的简洁。同时，HTML5 不是一种 XML 语言，其语法也很随意，下面从这两方面进行逐句分析。

第一行代码如下：

```
<!DOCTYPE HTML>
```

不需要包括版本号，仅告诉浏览器需要一个 doctype 来触发标准模式，可谓简明扼要。

接下来说明文档的字符编码，否则将出现浏览器不能正确解析的情况。

```
<meta charset="utf-8">
```

同样也很简单，HTML5 不区分大小写，不需要标记结束符，不介意属性值是否加引号，即下列代码是等效的：

```
<meta charset="utf-8">
<META charset="utf-8" />
<META charset=utf-8>
```

在主体中，可以省略主体标记，直接编写需要显示的内容。虽然在编写代码时省略了<html>、<head>和<body>标记，但在浏览器进行解析时，将会自动进行添加。 但是，考虑到代码的可维护性，在编写代码时，应该尽量增加这些基本结构标签。



视频讲解

1.4.2　比较 HTML4 与 HTML5 文档结构

下面通过示例具体说明 HTML5 是如何使用全新的结构化标签设计网页的。



【示例 1】本例设计将页面分成上、中、下三部分：上面显示网站标题；中间分两部分，左侧为辅助栏，右侧显示网页正文内容；下面显示版权信息，如图 1.2 所示。使用 HTML4 构建文档基本结构如下：

```
<div id="header">[标题栏]</div>
<div id="aside">[侧边栏]</div>
<div id="article">[正文内容]</div>
<div id="footer">[页脚栏]</div>
```



图 1.2　简单的网页布局



示例效果

尽管上述代码不存在任何语法错误，也可以在 HTML5 中很好地解析，但该页面结构对于浏览器来说是不具有区分度的。对于不同的用户来说，ID 命名可能因人而异，这对浏览器来说，就无法辨别每个 div 元素在页面中的作用，因此也必然会影响其对页面的语义解析。

【示例 2】下面使用 HTML5 新增元素重新构建页面结构，明确定义每部分在页面中的作用。

```
<header>[标题栏]</header>
<aside>[侧边栏]</aside>
<article>[正文内容]</article>
<footer>[页脚栏]</footer>
```

虽然两段代码不一样，但比较上述两段代码，使用 HTML5 新增元素创建的页面代码更简洁、明晰。可以很容易看出，使用<div id="header">、<div id="aside">、<div id="article">和<div id="footer">这些标记元素没有任何语义，浏览器也不能根据标记的 ID 名称来推断它的作用，因为 ID 名称是随意变化的。

而 HTML5 新增元素 header，明确地告诉浏览器此处是页头，aide 元素用于构建页面辅助栏目，article 元素用于构建页面正文内容，footer 元素定义页脚注释内容。这样极大地提高了开发者的便利性和浏览器的解析效率。

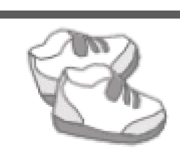
1.4.3　设计一个较详细的 HTML5 文档模板

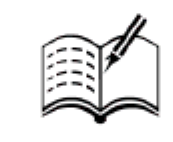


视频讲解

【示例 1】下面是一个简单的 HTML5 文档模板代码。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
```






```
<title>Hello HTML5</title>
</head>
<body>
</body>
</html>
```

HTML5 文档以<!DOCTYPE html>开头，这是一个文档类型声明，并且必须位于 HTML5 文档的第一行，它告诉浏览器当前文档的类型。

<html>标签是 HTML5 文档的根标签，位于<!DOCTYPE html>标签的下面。<html>标签支持 HTML5 全局属性和 manifest 属性。manifest 属性用于创建 HTML5 离线应用。

<head>标签是网页头部容器。位于<head>内部的元素可以包含脚本、样式表、元信息等。<head>标签也支持 HTML5 全局属性。

<meta>标签位于文档头部区域，不包含任何内容。使用<meta>的属性可以定义文档元信息，属性值以名/值对的形式设置。例如，<meta charset="utf-8" />定义了文档的字符编码，这里 charset 是<meta>标签的属性，"utf-8"是该属性的值。HTML5 中大部分标签都定义有属性，以扩展标签的功能。

<title>标签位于<head>标签内，定义网页文档的标题，显示在浏览器标题栏，或当网页被添加到收藏夹时作为默认标题使用，也方便搜索引擎抓取。因此，当写 HTML5 文档时一定要设置该标签。

<body>标签定义网页正文，文档的所有内容，如文本、超链接、图像、表格、列表等都包含在该标签中，并显示在页面中。



【示例 2】为了帮助读者更好地对 HTML5 文档有一个全局认识，也为了让读者初步了解复杂的 HTML5 文档代码，下面给出一个详细的、符合标准的 HTML5 文档模板，并进行详细注释。当然，在编写 HTML5 文档时，这些代码不是必需的，使用时可以根据需要酌情增删。

具体代码展示请查看示例源代码，或者扫码学习。

1.5　HTML5 API

HTML5 引入了很多新的 API 和扩展，并对一些现有的 API 进行了修改或删除。本节仅做选修，感兴趣的读者可扫码了解。

1.5.1　新增的 API



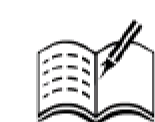
HTML5 是一种开放的 Web 标准，其技术内涵和 API 外延会随着 Web 的发展而不断丰富。目前 HTML5 主要包括下面这些功能。这些 API 可以与应用程序中引入的新元素一起使用。

详细内容请扫码阅读。

1.5.2　修改的 API



以下几点是基于 2 级 DOM HTML 的各种使用方式进行修改。详细内容请扫码阅读。



1.5.3 扩展 Document



2 级 DOM HTML 有一个 HTMLDocument 接口，继承于 Document，在此基础上，HTML5 提供了一些特殊的成员和功能。

详细内容请扫码阅读。



1.5.4 扩展 HTMLElement



HTMLElemen 接口也在 HTML 范围下获得了几个扩展。详细内容请扫码阅读。

1.5.5 扩展 DOM HTML



在 2 级 DOM HTML 接口中新增一些其他功能的扩展。详细内容请扫码阅读。

1.5.6 弃用的 API



在 HTML5 中，一些 API 已经被移除，或者标记为过时。详细内容请扫码阅读。

1.6 在 线 练 习

练习设计 HTML5 文档的基本方法。



第2章

HTML5 新增元素和文档结构

为了使文档结构更加清晰、明确、容易阅读，HTML5 增加了很多新的元素。本章将详细介绍这些新增的元素，涉及它们的定义、使用方法以及演示示例，最后再介绍如何应用这些新元素设计崭新的 HTML5 文档。

【学习重点】

- 正确使用 HTML5 结构元素。
- 正确使用 HTML5 语义元素。
- 设计符合语义的 HTML5 文档结构。





2.1　HTML5 元素

HTML5 包含一百多个标签，大部分继承自 HTML4，同时新增 30 个标签，简单介绍如下。

2.1.1　新元素分类

根据现有的规范，把 HTML5 新增元素按优先级分为 4 大类，简单说明如下。

- ☑　结构性元素。
- ☑　级块性元素。
- ☑　行内语义性元素。
- ☑　交互性元素。

详细说明请扫码了解。



线 上 阅 读

2.1.2　废除的元素

HTML5 废除了 HTML4 中部分过时的元素，例如：

- ☑　使用 CSS 替代的元素。
- ☑　弃用 frame 框架。
- ☑　部分浏览器支持的私有元素。

详细说明请扫码了解。



线 上 阅 读

2.2　设计新的文档结构

HTML5 新增多个结构化元素，以方便用户创建更友好的页面主体框架，下面来详细学习。

2.2.1　article——文章块



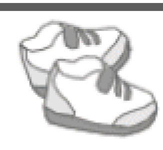
视 频 讲 解

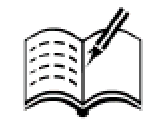
article 表示文章，用来标识页面中一块完整的、独立的、可以被转发的内容。例如，报纸文章、论坛帖子、用户评论、博客条目等。


提示：一些交互式小部件或小工具，或任何其他可独立的内容，原则上都可以作为 article 块，如日期选择器组件，但这些内容不是 HTML5 新增 article 元素的主要目的，故不要滥用。

【示例 1】 article 内容块通常包含标题，放在 header 元素里面，有时还包含 footer，定义附加信息。下面示例演示了如何使用 article 设计一篇新闻稿。

```
<article>
    <header>
        <h1>首届 Web 高层论坛（Web Executive Forum）将于 2017 年 11 月在美国旧金山举行 </h1>
        <time pubdate="pubdate">2017 年 9 月 26 日消息</time>
    </header>
    <p>W3C 将于 2017 年 11 月 8 日在美国加州旧金山举行首届 W3C Web 高层论坛(Web Executive Forum),
```





Note

```
来自支付宝（Alipay）、美国运通（American Express）、彭博（Bloomberg）、哈曼（HARMAN）、谷歌（Google）、英特尔（Intel）、Mozilla、三星（Samsung）、南内华达地区交通局（Southern Nevada Regional Transportation Agency）、悉尼大学（University of Sydney）、Worldpay、Yubico 等机构的代表将与 W3C 的发明人、W3C 理事长 Tim Berners Lee 一起，探讨 Web 的技术趋势及对行业产业的影响。这是 W3C 首次举办此类论坛，论坛将与 W3C TPAC 2017 会议同期举行。</p>
        <footer>
            <p>来自<a href="http://www.chinaw3c.org/archives/1980/" target="_blank">W3C 中国</a></p>
        </footer>
    </article>
```

上面示例是一篇互联网新闻的文章，在 header 元素中嵌入了文章的标题部分，在这部分中，文章的标题被镶嵌在 h1 元素中，文章的发表日期镶嵌在 time 元素中。在标题下部的 p 元素中，嵌入了一大段文章的正文，在结尾处的 footer 元素中，作为脚注，嵌入了文章的来源。整个文章块的内容相对比较独立、完整，因此对这部分内容使用了 article 元素。

【示例 2】 article 元素可以嵌套使用，原则上内层的内容要与外层的内容相关联。例如，在一篇互联网新闻中，针对该新闻的相关评论就可以使用嵌套 article 元素设计，用来呈现评论的 article 元素被包含在外层 article 元素里面。

```
<article>
    <header>[省略]</header>
    <p>[请参考示例 1]</p>
    <footer>…</footer>
    <section>
        <h2>评论</h2>
        <article>
            <header>
                <h3>网友昵称 1</h3>
                <p>
                    <time pubdate datetime="2017-9-26 19:40-08:00"> 1 小时前 </time>
                </p>
            </header>
            <p>ok</p>
        </article>
        <article>[参考第一条评论的结构]</article>
        <article>[每条评论作为一个相对独立的内容块]</article>
        <article>[内层 article 块与外层 article 块相关联]</article>
        …
    </section>
</article>
```

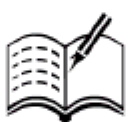
上面示例的内容比示例 1 的内容更加丰富，它添加了评论内容。具体来说，section 把正文与评论部分进行了区分，在 section 元素中嵌入了评论的内容，评论中每一个人的评论相对来说又是比较独立、完整的，因此每条评论都使用一个 article，在评论中又可以分为标题和评论内容两部分，分别放在 header 元素与 p 元素中。


视频讲解

2.2.2　section——区块

section 表示区块，用于标识文档中的节，在页面上多对内容进行分区。例如，章节、页眉、页脚或文档中的其他部分。





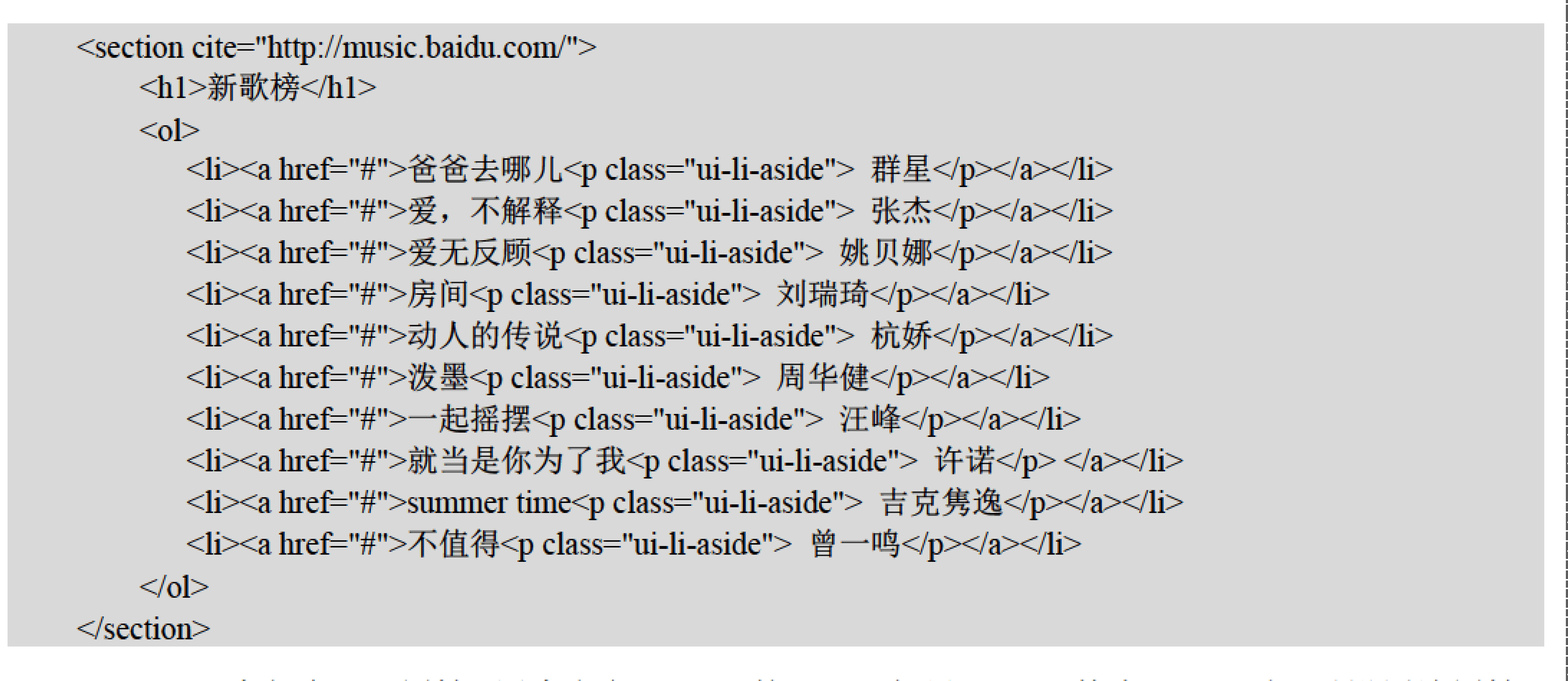
Note

【辨析】

div 元素也可以用来对页面进行分区，但 section 元素并非一个普通的容器。当一个容器需要被直接定义样式或通过脚本定义行为时，推荐使用 div，而非 section 元素。

div 元素关注结构的独立性，而 section 元素关注内容的独立性，section 元素包含的内容可以单独存储到数据库中，或输出到 Word 文档中。

【示例 1】一个 section 区块通常由标题和内容组成。下面示例使用 section 元素包裹排行版的内容，作为一个独立的内容块进行定义。



```
<section cite="http://music.baidu.com/">
    <h1>新歌榜</h1>
    <ol>
        <li><a href="#">爸爸去哪儿<p class="ui-li-aside"> 群星</p></a></li>
        <li><a href="#">爱，不解释<p class="ui-li-aside"> 张杰</p></a></li>
        <li><a href="#">爱无反顾<p class="ui-li-aside"> 姚贝娜</p></a></li>
        <li><a href="#">房间<p class="ui-li-aside"> 刘瑞琦</p></a></li>
        <li><a href="#">动人的传说<p class="ui-li-aside"> 杭娇</p></a></li>
        <li><a href="#">泼墨<p class="ui-li-aside"> 周华健</p></a></li>
        <li><a href="#">一起摇摆<p class="ui-li-aside"> 汪峰</p></a></li>
        <li><a href="#">就当是你为了我<p class="ui-li-aside"> 许诺</p> </a></li>
        <li><a href="#">summer time<p class="ui-li-aside"> 吉克隽逸</p></a></li>
        <li><a href="#">不值得<p class="ui-li-aside"> 曾一鸣</p></a></li>
    </ol>
</section>
```

section 元素包含 cite 属性，用来定义 section 的 URL。如果 section 摘自 Web，则可以设置该属性。

【辨析】

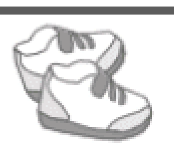
article 和 section 都是 HTML5 新增的元素，它们都用来区分不同内容，用法也相似，从语义角度分析两者区分很大。

- ☑ article 代表文档、页面或者应用程序中独立、完整的可以被外部引用的内容。因为 article 是一段独立的内容，所以 article 通常包含 header 和 footer 结构。
- ☑ section 用于对网站或者应用程序中页面上的内容进行分块。一个 section 通常由内容和标题组成。因此，需要包含一个标题，一般不用包含 header 或者 footer 结构。

通常使用 section 元素为那些有标题的内容进行分段，类似文章分段操作。相邻的 section 内容应当是相关的，而不像 article 之间各自独立。

【示例 2】下面示例混用 article 和 section 元素，从语义上比较两者不同。article 内容强调独立性、完整性，section 内容强调相关性。

```
<article>
    <header>
        <h1>蝶恋花</h1>
        <h2>晏殊</h2>
    </header>
    <p>槛菊愁烟兰泣露，罗幕轻寒，燕子双飞去。明月不谙离恨苦，斜光到晓穿朱户。</p>
    <p>昨夜西风凋碧树，独上高楼，望尽天涯路。欲寄彩笺兼尺素，山长水阔知何处。</p>
    <section>
        <h2>解析</h2>
        <article>
            <h3>注释</h3>
```



Note

```
                <p>槛：栏杆。</p>
                <p>罗幕：丝罗的帷幕，富贵人家所用。</p>
                <p>朱户：犹言朱门，指大户人家。</p>
                <p>尺素：书信的代称。</p>
            </article>
            <article>
                <h3>评析</h3>
                <p>此词经疏澹的笔墨、温婉的格调、谨严的章法，传达出作者的暮秋怀人之情。 </p>
                <p>上片由苑中景物起笔，下片写登楼望远。以无可奈何的怅问作结，给人情也悠悠、恨也悠悠之感。 </p>
            </article>
        </section>
    </article>
```

【追问】

既然 article、section 是用来划分区域的，又是 HTML5 的新元素，那么是否可以用 article、section 取代 div 来布局网页呢？

答案是否定的，div 的用处就是布局网页，划分大的区域，所以我们习惯性地把 div 当成了一个容器。而 HTML5 改变了这种用法，它让 div 的工作更纯正。div 就是用来布局的，在不同的内容块中，我们按照需求添加 article、section 等内容块，并且显示其中的内容，这样才是对这些元素的合理使用。

因此，在使用 section 元素时应该注意几个问题：

☑ 不要将 section 元素当作设置样式的结构容器，对于此类操作应该使用 div 元素实现。

☑ 如果 article、aside 或 nav 元素更符合语义使用条件，不要首选使用 section 元素。

☑ 不要为没有标题的内容区块使用 section 元素。

【补充】

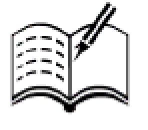
使用 HTML5 大纲工具（http://gsnedders.html5.org/outliner/）来检查页面中是否有没标题的 section，如果使用该工具进行检查后，发现某个 section 的说明中有“untitiled section”（没有标题的 section）文字，这个 section 就有可能使用不当，但是 nav 和 aside 元素没有标题是合理的。

【示例 3】下面示例进一步演示了 article 和 section 混用的情景。

```
    <article>
        <h1>W3C</h1>
        <p>万维网联盟（World Wide Web Consortium，W3C），又称 W3C 理事会。1994 年 10 月在麻省理工学院计算机科学实验室成立。建立者是万维网的发明者蒂姆&middot;伯纳斯-李。</p>
        <section>
            <h2>CSS</h2>
            <p>全称 Cascading Style Sheet，级联样式表，通常又称为“风格样式表（Style Sheet)”，它是用来进行网页风格设计的。</p>
        </section>
        <section>
            <h2>HTML</h2>
            <p>全称 Hypertext Markup Language，超文本标记语言，用于描述网页文档的一种标记语言。</p>
        </section>
    </article>
```

在上面示例中，首先可以看到整个版块是一段独立的、完整的内容，因此使用 article 元素标识。该内容是一篇关于 W3C 的简介，该文章分为 3 段，每一段都有一个独立的标题，因此使用了两个 section 元素区分。





【追问】

为什么没有对第一段使用 section 元素呢？

其实是可以使用的，但是由于其结构比较清晰，浏览器能够识别第一段内容在一个 section 内，所以也可以将第一个 section 元素省略，但是如果第一个 section 元素里还要包含子 section 元素或子 article 元素，那么就必须标识 section 元素。

【示例 4】 下面是一个包含 article 元素的 section 元素示例。

```
<section>
    <h1>W3C</h1>
    <article>
        <h2>CSS</h2>
        <p>全称 Cascading Style Sheet，级联样式表，通常又称为“风格样式表（Style Sheet）”，它是用来进行网页风格设计的。</p>
    </article>
    <h2>HTML</h2>
    <p>全称 Hypertext Markup Language，超文本标记语言，用于描述网页文档的一种标记语言。</p>
</section>
```

这个示例比示例 3 复杂了一些。首先，它是一篇文章中的一段，因此没有使用 article 元素。但是，在这一段中有几块独立的内容，所以嵌入了几个独立的 article 元素。

在 HTML5 中，article 可以是一种特殊功能的 section 元素，它比 section 元素更强调独立性。即 section 元素强调分段或分块，而 article 强调独立性。具体来说，如果一块内容相对来说比较独立、完整，应该使用 article 元素，但是如果想将一块内容分成几段，应该使用 section 元素。

在 HTML5 中，div 变成了一种容器，当应用 CSS 样式的时候，可以对这个容器进行一个总体的 CSS 样式的套用。因此，可以将页面的所有从属部分，如导航条、菜单、版权说明等，包含在一个统一的页面结构中，以便统一使用 CSS 样式来进行装饰。

2.2.3　nav——导航条



nav 表示导航条，用来标识页面导航的链接组。一个页面中可以拥有多个 nav，作为页面整体或不同部分的导航。具体应用场景如下：

- ☑ 主菜单导航。一般网站都设置有不同层级的导航条，其作用是在站内快速切换，如主菜单、置顶导航条、主导航图标等。
- ☑ 侧边栏导航。现在主流博客网站及商品网站上都有侧边栏导航，其作用是将页面从当前文章或当前商品跳转到相关文章或商品页面上去。
- ☑ 页内导航。就是页内锚点链接，其作用是在本页面几个主要的组成部分之间进行跳转。
- ☑ 翻页操作。翻页操作是指在多个页面的前后页或博客网站的前后篇文章滚动。

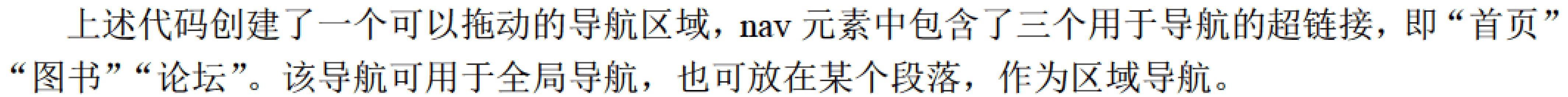
并不是所有的链接组都要被放进 nav 中，只需要将主要的、基本的链接组放进 nav 元素即可。例如，在页脚中通常会有一组链接，包括服务条款、首页、版权声明等，这时使用 footer 元素最恰当。

【示例 1】 在 HTML5 中，只要是导航性质的链接，我们就可以很方便地将其放入 nav 元素中。该元素可以在一个文档中多次出现，作为页面或部分区域的导航。

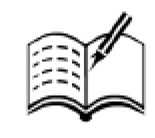
```
<nav draggable="true">
    <a href="index.html">首页</a>
    <a href="book.html">图书</a>
```



```
        <a href="bbs.html">论坛</a>
</nav>
```



上述代码创建了一个可以拖动的导航区域，nav 元素中包含了三个用于导航的超链接，即“首页”“图书”“论坛”。该导航可用于全局导航，也可放在某个段落，作为区域导航。



Note

【示例 2】下面示例由多部分组成，每部分都有链接，但只将最主要的链接放入 nav 元素中。

```
<h1>技术资料</h1>
<nav>
    <ul>
        <li><a href="/">主页</a></li>
        <li><a href="/blog">博客</a></li>
    </ul>
</nav>
<article>
    <header>
        <h1>HTML5+CSS3</h1>
        <nav>
            <ul>
                <li><a href="#HTML5">HTML5</a></li>
                <li><a href="#CSS3">CSS3</a></li>
            </ul>
        </nav>
    </header>
    <section id="HTML5">
        <h1>HTML5</h1>
        <p>HTML5 特性说明</p>
    </section>
    <section id="CSS3">
        <h1>CSS3</h1>
        <p>CSS3 特性说明</p>
    </section>
    <footer>
        <p> <a href="?edit">编辑</a> | <a href="?delete">删除</a> | <a href="?add">添加</a> </p>
    </footer>
</article>
<footer>
    <p><small>版权信息</small></p>
</footer>
```

在这个例子中，第一个 nav 元素用于页面导航，将页面跳转到其他页面上去，如跳转到网站主页或博客页面；第二个 nav 元素放置在 article 元素中，表示在文章中进行导航。除此之外，nav 元素也可以用于其他所有你觉得重要的、基本的导航链接组中。

注意：不要用 menu 元素代替 nav 元素。menu 主要用在一系列交互命令的菜单上，如快捷菜单。

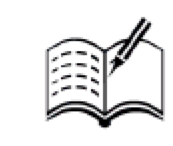
2.2.4 aside——辅助栏



视频讲解

aside 表示侧边，用来标识所处内容之外的内容。aside 内容应该与所处的附近内容相关。例如，当前页面或文章的附属信息部分，它可以包含与当前页面或主要内容相关的引用、侧边广告、导航条，以及其他类似的有别于主要内容的部分。





aside 元素主要有两种用法：

（1）作为主体内容的附属信息部分，包含在 article 中，aside 内容可以是与当前内容有关的参考资料、名词解释等。

【示例 1】下面示例设计一篇文章，文章标题放在 header 中，在 header 后面将所有关于文章的部分放在了一个 article 中，将文章正文放在一个 p 元素中。该文章包含一个名词注释的附属部分，因此在正文下面放置了一个 aside 元素，用来存放名词解释的内容。

```
<header>
    <h1>HTML5</h1>
</header>
<article>
    <h1>HTML5 历史</h1>
    <p>HTML5 草案的前身名为 Web Applications 1.0，于 2004 年被 WHATWG 提出，于 2007 年被 W3C 接纳，并成立了新的 HTML 工作团队。HTML5 的第一份正式草案已于 2008 年 1 月 22 日公布。2014 年 10 月 28 日，W3C 的 HTML 工作组正式发布了 HTML5 的官方推荐标准。</p>
    <aside>
        <h1>名词解释</h1>
        <dl>
            <dt>WHATWG</dt>
            <dd>Web Hypertext Application Technology Working Group,HTML 工作开发组的简称，目前与 W3C 组织同时研发 HTML5。</dd>
        </dl>
        <dl>
            <dt>W3C</dt>
            <dd>World Wide Web Consortium，万维网联盟，万维网联盟是国际著名的标准化组织。1994 年成立后，至今已发布近百项万维网相关的标准，对万维网发展做出了杰出的贡献。</dd>
        </dl>
    </aside>
</article>
```

aside 被放置在 article 内部，因此引擎将这个 aside 内容理解为与 article 内容相关联的。

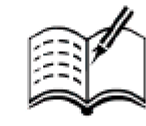
（2）作为页面或站点辅助功能部分，在 article 之外使用。最典型的形式是侧边栏，其中的内容可以是友情链接、最新文章列表、最新评论列表、历史存档、日历等。

【示例 2】下面代码使用 aside 元素为个人博客添加一个友情链接辅助版块。

```
<aside>
    <nav>
        <h2>友情链接</h2>
        <ul>
            <li> <a href="#">网站 1</a></li>
            <li> <a href="#">网站 2</a></li>
            <li> <a href="#">网站 3</a></li>
        </ul>
    </nav>
</aside>
```

友情链接在博客网站中比较常见，一般放在左右两侧的边栏中，因此可以使用 aside 来实现，但是这个版块又具有导航作用，因此嵌套了一个 nav 元素，该侧边栏的标题是“友情链接”，放在了 h2 元素中，在标题之后使用了一个 ul 列表，用来存放具体的导航链接列表。


视频讲解


Note

2.2.5 main——主要区域

main 表示主要，用来标识网页中的主要内容。main 内容对于文档来说应当是唯一的，它不应包含在网页中重复出现的内容，如侧栏、导航栏、版权信息、站点标志或搜索表单等。

简单来说，在一个页面中，不能出现一个以上的 main 元素。main 元素不能被包含在 article、aside、footer、header 或 nav 中。

提示：由于 main 元素不对页面内容进行分区或分块，所以不会对网页大纲产生影响。

【示例】下面示例使用 main 元素包含页面主要区域，这样更有利于网页内容的语义分区，同时搜索引擎也能够主动抓取主要信息，避免被次要信息干扰。

```
<header>
    <nav>
        <ul>
            <li><a href="#">首页</a></li>
            <li><a href="#">站内新闻</a></li>
            <li><a href="#">站外新闻</a></li>
        </ul>
    </nav>
</header>
<main>
    <h1>站内新闻</h1>
    <nav>
        <ul>
            <li><a href="#">HTML5</a></li>
            <li><a href="#">CSS3</a></li>
            <li><a href="#">JavaScript</a></li>
        </ul>
    </nav>
    <H2 id="web">W3C</H2>
    <h3>W3C 中国区会员沙龙在北京航空航天大学举行</h3>
    <p>2017 年 9 月 14 日，W3C 在北京航空航天大学举办了中国区会员沙龙活动，向到会的中国区会员代表介绍 W3C 目前标准工作进展及计划，并提供一个新老朋友参与 W3C 及其他相关话题问答与互动讨论的交流平台。</p>
    <h2 id="new">最新新闻</h2>
    <ul>
        <li>W3C 发布 ODRL 信息模型、ODRL 词汇表及表达两份候选推荐标准 征集参考实现及审阅意见</li>
        <li>W3C 技术研讨会：Web 虚拟现实编著—机遇与挑战</li>
        <li>W3C 发布核心无障碍 API 映射（Core-AAM）1.1 版候选推荐标准 征集参考实现</li>
    </ul>
    <h2 id="blog">W3C 官方博客</h2>
    <ul>
        <li>W3C 启动 WebAssembly 工作组</li>
        <li>W3C 数据的未来方向</li>
        <li>W3C 数字出版主要进展</li>
    </ul>
</main>
<footer>本站由北京航空航天大学(W3C/Beihang)维护 京 ICP 备 05004617-3 文保网安备案号
```





视频讲解



Note

2.2.6　header——标题栏

header 表示页眉，用来标识页面标题栏。header 元素是一种具有引导和导航作用的结构元素，通常用来放置整个页面或者一个内容块的标题。

header 也可以包含其他内容，如数据表格、表单或相关的 LOGO 信息，一般整个页面的标题应该放在页面的前面。

【示例 1】在一个网页内可以多次使用 header 元素，下面示例显示为每个内容区块添加一个 header。

```
<header>
    <h1>网页标题</h1>
</header>
<article>
    <header>
        <h1>文章标题</h1>
    </header>
    <p>文章正文</p>
</article>
```

在 HTML5 中， header 内部可以包含 h1～h6 元素，也可以包含 hgroup、table、form、nav 等元素，只要应该显示在头部区域的标签，都可以包含在 header 元素中。

【示例 2】下面示例是个人博客首页的头部区域，整个头部内容都放在 header 元素中。

```
<header>
    <hgroup>
        <h1>LOGO</h1>
        <a href="#">[URL]</a> <a href="#">[订阅]</a> <a href="#">[手机订阅]</a>
    </hgroup>
    <nav>
        <ul>
            <li>首页</li>
            <li><a href="#">目录</a></li>
            <li><a href="#">社区</a></li>
            <li><a href="#">微博我</a></li>
        </ul>
    </nav>
</header>
```

视频讲解

2.2.7　hgroup——标题组

hgroup 表示标题分组，用来为标题或子标题进行分组。通常 hgroup 与 h1～h6 元素组合使用，一个内容块中的标题及其子标题可以通过 hrgoup 组成一组。但是，如果文章只有一个主标题，则不需要 hgroup 元素。

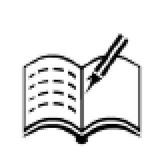
注意：hgroup 元素在 HTML5.1 版中被废除，不再建议使用。替代方案请扫码了解。



线上阅读

【示例】下面示例显示如何使用 hgroup 元素把主标题、副标题和标题说明进行分组，以便让引擎更容易识别标题块。

```
<article>
    <header>
        <hgroup>
            <h1>首届 Web 高层论坛将于 2017 年 11 月在美国旧金山举行</h1>
            <h2>September 26, 2017</h2>
            <h3>国际新闻,TPAC 及 AC,博客文章,技术活动 </h3>
        </hgroup>
    </header>
    <p>本次论坛的议程包括一系列圆桌讨论（Panel Discussion）和高端对话：</p>
    <ul>
        <li>Web 支付的未来（Future of Payments on the Web）</li>
        <li>网联汽车、城市和 Web（Connected Cars、Cities and Web）</li>
        <li>Web 新兴技术（Emerging Technologies）</li>
        <li>对话：Web 的未来，嘉宾：Brad Stone （彭博）、Sir Tim Berners Lee （W3C） </li>
    </ul>
    </p>
</article>
```





视 频 讲 解

2.2.8 footer——页脚栏

footer 表示脚注，用来标识文档或节的页脚。footer 元素应当含有其包含元素的信息。例如，页脚通常包含文档的作者、版权信息、使用条款链接、联系信息等。

【示例 1】在 HTML4 中，一般使用<div id="footer">包含页脚信息，现在使用 footer 元素来替代，更富有语义。下面示例使用 footer 元素为页面添加版权信息栏目。

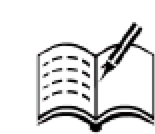
```
<article>
    …
</article>
<footer>
    <ul>
        <li>关于</li>
        <li>导航</li>
        <li>联系</li>
    </ul>
</footer>
```

【示例 2】在一个页面中，可以使用多个 footer 元素。同时，可以为 article 或 section 内容添加 footer。下面示例分别在 article、section 和 body 区域内添加 footer 信息。

```
<header>
    <h1>网页标题</h1>
</header>
<article>
    <h2>文章标题</h2>
    <p>文章内容正文</p>
    <footer>注释</footer>
</article>
<section>
    <h2>段落标题</h2>
    <p>正文</p>
```



```
        <footer>段落标记</footer>
    </section>
    <footer>网页版权信息</footer>
```



2.3　设计新的语义信息

HTML5 不仅增加了很多结构元素，也增加了很多实用语义元素，下面来详细学习。

2.3.1　address——联系信息



address 元素用来在文档中定义联系信息，包括文档作者或文档编辑者名称、电子邮箱、真实地址、电话号码等。

【示例 1】address 元素的用途不仅仅是描述电子邮箱或真实地址，还可以描述与文档相关的联系人的所有联系信息。下面代码展示了博客侧栏中的一些技术参考网站网址链接。

```
<address>
    <a href="http://www.w3.org/">W3C</a>
    <a href="http://www.whatwg.org/">WHATWG</a>
    <a href="http://www.mhtml5.com/">HTML5 研究小组</a>
</address>
```

【示例 2】也可以把 footer 元素、time 元素与 address 元素结合起来使用，以实现设计一个比较复杂的版块结构。

```
<footer>
    <section>
        <address>
        <a title="作者：MDN" href="https://developer.mozilla.org/zh-CN/docs/Web/Guide/HTML/HTML5">
HTML5 - Web 开发者指南</a>
        </address>
        <p> 发布于：
            <time datetime="2017-6-1">2017 年 6 月 1 日</time>
        </p>
    </section>
</footer>
```

这个示例把博客文章的作者、博客的主页链接作为作者信息放在了 address 元素中，把文章发表日期放在了 time 元素中，把这个 address 元素与 time 元素中的总体内容作为脚注信息放在了 footer 元素中。

2.3.2　time——显示时间



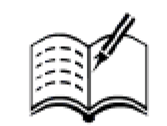
time 元素定义公历的时间（24 小时制）或日期，时间和时区偏移是可选的。

该元素能够以机器可读的方式对日期和时间进行编码。例如，浏览器能够把生日提醒或排定的事件添加到用户日程表中，搜索引擎也能够生成更智能的搜索结果。

【示例 1】下面示例演示如何定义时间和日期。



```
<p>我们每天早上 <time>9:00</time> 打卡上班。</p>
<p>我在 <time datetime="2018-02-14">情人节</time> 有个约会。</p>
```



Note

【拓展】

time 元素定义了两个属性，简单说明如下：

☑ datetime：规定日期和时间的格式。否则，由元素的内容给定日期和时间。

☑ pubdate：定义 time 元素中的日期和时间是文档或 article 内容的发布日期。

【示例 2】time 元素可以定义很多格式的日期和时间。

```
<time datetime="2017-11-13">2017 年 11 月 13 日</time>
<time datetime="2017-11-13">11 月 13 日</time>
<time datetime="2017-11-13">我的生日</time>
<time datetime="2017-11-13T20:00">我生日的晚上 8 点</time>
<time datetime="2017-11-13T20:00Z">我生日的晚上 8 点</time>
<time datetime="2017-11-13T20:00+09:00">我生日的晚上 8 点的美国时间</time>
```

浏览器通过 datetime 属性获取 time 的时间，而 time 开始标记与结束标记中间的部分是显示在网页上的。datetime 属性中日期与时间之间要用“T”文字分隔，“T”表示时间。

注意： 倒数第二行，时间加上 Z 文字表示给机器编码时使用 UTC 标准时间，倒数第一行则加上了时差，表示向机器编码另一地区时间，如果是编码本地时间，则不需要添加时差。

pubdate 属性是一个可选的布尔值属性，它可以用在 article 元素中的 time 元素上，意思是 time 元素代表了文章（article 元素的内容）或整个网页的发布日期。注意，HTML5 新标准不再支持 pubdate 属性。

【示例 3】下面示例使用 pubdate 属性为文档添加引擎检索的发布日期。

```
<article>
    <header>
        <h1>首届 Web 高层论坛（Web Executive Forum）将于 2017 年 11 月在美国旧金山举行 </h1>
        <p>发布日期<time datetime="2017-9-26" pubdate>2017 年 9 月 26 日消息</time></p>
    </header>
    <p>…</p>
    <footer>
        <p>来自<a href="http://www.chinaw3c.org/archives/1980/" target="_blank">W3C 中国</a></p>
    </footer>
</article>
```

由于 time 元素不仅仅表示发布时间，而且还可以表示其他用途的时间，如通知、约会等。

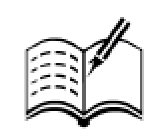
【示例 4】为了避免引擎误解发布日期，使用 pubdate 属性可以显式告诉引擎文章中哪个是真正的发布时间。

```
<article>
    <header>
        <h1>首届 Web 高层论坛（Web Executive Forum）将于 2017 年 11 月在美国旧金山举行 </h1>
        <p>发布日期<time datetime="2017-9-26" pubdate>2017 年 9 月 26 日消息</time></p>
        <p>关于<time datetime=2017-10-1>10 月 1 日</time>参会紧急通知</p>
    </header>
    <p>…</p>
    <footer>
```



```
        <p>来自<a href="http://www.chinaw3c.org/archives/1980/" target="_blank">W3C 中国</a></p>
    </footer>
</article>
```

在这个例子中，有两个 time 元素，分别定义了两个日期：紧急通知日期和发布日期。由于都使用了 time 元素，所以需要使用 pubdate 属性表明哪个 time 元素代表了新闻的发布日期。



Note



视频讲解

2.3.3　figure 和 figcaption——流媒体

figure 元素可以定义独立的流内容，如图像、图表、照片、代码等。figure 元素的内容应该与主内容相关，但如果被删除，则不会对文档产生影响。

figcaption 元素表示 figure 元素的标题，它从属于 figure 元素，必须书写在 figure 元素内部，可以书写在 figure 元素内的其他从属元素的前面或后面。一个 figure 元素内最多只允许放置一个 figcaption 元素，但允许放置多个其他元素。

【示例 1】下面示例设计一个不带标题的 figure 元素。

```
<figure>
    <img src="images/1.jpg" width="400" alt="学生">
</figure>
```

【示例 2】下面示例设计一个带标题的 figure 元素，标题使用 figcaption 元素定义。

```
<figure>
    <img src="images/1.jpg" width="400" alt="学生">
    <figcaption>大学生</figcaption>
</figure>
```

【示例 3】下面示例设计为多幅图片设计使用同一个标题，演示效果如图 2.1 所示。

```
<figure>
    <img src="images/1.jpg" height="150" alt="学生">
    <img src="images/2.jpg" height="150" alt="学生">
    <img src="images/3.jpg" height="150" alt="学生">
    <figcaption>大学生</figcaption>
</figure>
```



图 2.1　多个图片使用同一个标题



视频讲解

2.3.4　details 和 summary——详细内容

details 元素用于描述文档或文档某个部分的细节，被 details 标识的内容可以展开或收缩显示。details 可以包含文字、表单、插件或表格等任何超文本信息。

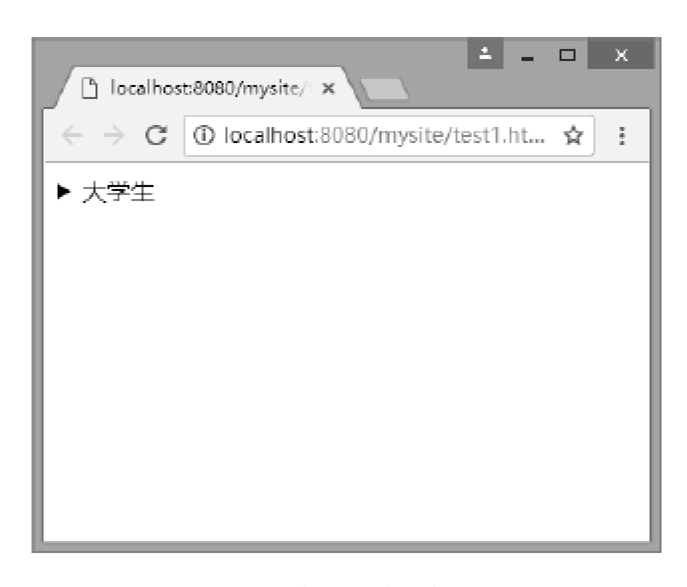
details 包含一个 open 属性，取值为布尔值，当该属性值为 true 时，其包含的子元素应该展开显

示；当该属性值为 false 时，其包含的子元素应该收缩起来不显示。open 属性的默认值为 false，页面打开时，其内部子元素处于收缩状态。

summary 元素为 details 的子元素，可以为 details 定义标题。标题是可见的，用户单击标题时，会显示出 details 包含的信息。

【示例 1】下面示例使用 details 和 summary 元素设计折叠面板效果，演示如图 2.2 所示。

```
<details>
    <summary>大学生</summary>
    <figure>
        <img src="images/1.jpg" width="300" alt="学生">
    </figure>
</details>
```




（a）默认收缩显示



（b）单击展开显示

图 2.2　使用 details 和 summary 元素设计折叠面板

【示例 2】如果 details 元素内没有 summary 元素，浏览器会提供默认文字以供单击，例如，“details”或某些本地化文字，如“详细信息”，浏览器也提供一个诸如上下箭头一类的图标，标示该区域可以展开或收缩，效果如图 2.3 所示。

```
<details>
    <figure>
        <img src="images/1.jpg" width="300" alt="学生">
    </figure>
</details>
```

当 details 元素的状态从展开切换为收缩，或从收缩切换为展开时，均触发 toggle 事件。

【示例 3】下面示例设计当用户切换 details 元素显示或隐藏显示时，取消 summary 元素轮廓效果，并给 details 元素包含的内容加一个边框，效果如图 2.4 所示。

```
<details id="detail">
    <summary>大学生</summary>
    <figure id="info">
        <img src="images/1.jpg" width="300" alt="学生">
    </figure>
</details>
<script>
var detail=document.getElementById('detail');
var info=document.getElementById('info');
var summary=detail.getElementsByTagName('summary')[0];
```



```
detail.ontoggle = function(){
        if(this.open){
            info.style.border="solid 1px red";
            summary.style.outline="none";
        }else{
            info.style.border="solid 1px #fff";
        }
}
</script>
```

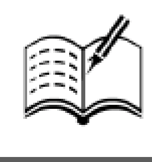




图 2.3　使用 details 设计折叠面板



图 2.4　取消标题轮廓线

提示： 目前，IE 浏览器暂不支持 details 和 summary 元素。

2.3.5　mark——记号文本


视 频 讲 解

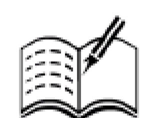
mark 元素用来定义带有记号的文本，它表示页面中需要临时高亮显示的信息，对于当前浏览者来说具有提示作用。例如，在网页中检索某个关键词时，呈现匹配的检索结果，现在很多搜索引擎都用类似方法实现了 mark 元素的功能。

【示例 1】 下面示例使用 mark 元素高亮显示“HTML5”关键词，演示效果如图 2.5 所示。

```
<p>
2014 年 10 月 28 日，W3C 的 HTML 工作组正式发布了 HTML5 的正式推荐标准（W3C Recommendation）。W3C 在美国圣克拉拉举行的 W3C 技术大会及顾问委员会会议（TPAC 2014）上宣布了这一消息。 HTML5 是万维网的核心语言——可扩展标记语言的第 5 版。在这一版本中，增加了支持 Web 应用开发者的许多新特性，以及更符合开发者使用习惯的新元素，并重点关注定义清晰的、一致的准则，以确保 Web 应用和内容在不同用户代理（浏览器）中的互操作性。HTML5 是构建开放 Web 平台的核心。
</p>
<script>
var ps = document.getElementsByTagName("p");
Array.prototype.forEach.call(ps,function(p){
    p.innerHTML = p.innerHTML.replace(/HTML5/g, function(html){
        return   "<mark>"+ html +"</mark>"
    })
})
</script>
```

在上面脚本中，获取页面中所有正文文本，然后使用数组对象的 forEach()方法迭代每个 p 元素，使用字符串对象的 replace()方法，通过正则表达式匹配到正文中所有的“HTML5”关键词，把它替换为“<mark> HTML5</mark>”的 HTML 字符串进行显示。





Note

【示例 2】mark 元素还可以用于标记引文，为特殊目的把原文作者没有重点强调的内容标记出来。下面示例使用 mark 元素标记唐诗中韵脚字，方便浏览者浏览，效果如图 2.6 所示。

```
<article>
    <h2>静夜思 </h2>
    <h3>李白</h3>
    <p>床前明月<mark>光</mark>，疑是地上<mark>霜</mark>。</p>
    <p>举头望明月，低头思故<mark>乡</mark>。</p>
</article>
```



图 2.5　使用 mark 元素高亮显示关键词



图 2.6　使用 mark 元素高亮显示韵脚

【辨析】

在 HTML4 中，用户习惯使用 em 或 strong 元素来突出显示文字，但是 mark 元素的作用与这两个元素的作用是有区别的，不能混用。

mark 元素的标记目的与原文作者无关，或者说它不是被原文作者用来标示文字的，而是后来被引用时添加上去的。它的目的是吸引当前用户的注意力，供用户参考，希望能够对用户有帮助。而 strong 是原文作者用来强调一段文字的重要性的，如错误信息等，em 元素是作者为了突出文章重点文字而使用的。



视频讲解

2.3.6　progress——进度条

progress 元素定义任务的进度或进程。这个进度可以是不确定的，表示进度正在进行，但不清楚还有多少进度没有完成，也可以用 0 到某个最大数字（如 100）之间的数字来表示进度完成情况。一般与 JavaScript 一同使用，来动态显示任务的进度。

progress 元素包含两个属性，简单说明如下：

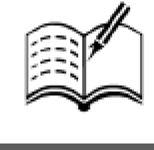
- ☑ max：规定任务一共需要多少工作。
- ☑ value：规定已经完成多少任务。

在设置属性的时候，value 和 max 属性只能指定为有效的浮点数，value 属性的值必须大于 0、小于或等于 max 属性的值，max 属性的值必须大于 0。

【示例】下面示例简单演示了如何使用 progress 元素，演示效果如图 2.7 所示。

```
<section>
    <p>百分比进度: <progress id="progress" max="100"><span>0</span>%</progress></p>
    <input type="button" onclick="click1()"    value="显示进度"/>
</section>
<script>
function click1(){
    var progress = document.getElementById('progress');
```






```
        progress.getElementsByTagName('span')[0].textContent ="0";
        for(var i=0;i<=100;i++)
            updateProgress(i);
    }
    function updateProgress(newValue){
        var progress = document.getElementById('progress');
        progress.value = newValue;
        progress.getElementsByTagName('span')[0].textContent = newValue;
    }
    </script>
```



图 2.7　使用 progress 元素

注意：progress 元素不适合用来表示度量衡，例如磁盘空间使用情况或查询结果。如需表示度量衡，应使用 meter 元素。

2.3.7　meter——度量



meter 元素定义已知范围或分数值内的标量测量。例如，磁盘用量、查询结果的相关性等。

注意，meter 元素不应用于指示进度，如果标记进度条，应使用 progress 元素。

meter 元素包含 7 个属性，简单说明如下：

☑ form：规定 <meter> 元素所属的一个或多个表单。
☑ high：规定被视作高的值的范围。如果小于 low 属性值，那么使用 low 属性值；如果大于 max 属性值，那么使用 max 属性值。
☑ low：规定被视作低的值的范围。必须小于或等于 high 属性值。如果小于 min 属性值，那么使用 min 属性值。
☑ max：规定范围的最大值。默认为 1，如果小于 min 属性值，那么使用 min 属性值。
☑ min：规定范围的最小值。默认为 0，不能小于 0。
☑ optimum：规定度量的优化值。必须在 min 和 max 属性值之间，可以大于 high 属性值。
☑ value：必需，规定度量的当前值。默认为 0，可以指定一个浮点小数值。

【示例】下面示例简单演示了如何使用 meter 元素，效果如图 2.8 所示。

```
<meter value="3" min="0" max="10">十分之三</meter>
<meter value="0.6">60%</meter>
```

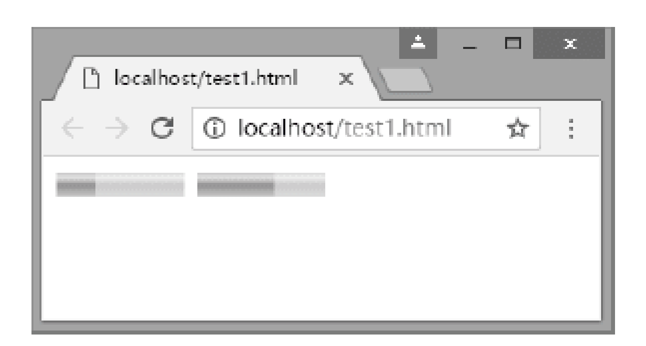


图 2.8　使用 meter 元素





视频讲解

Note

提示：目前，IE 浏览器暂不支持该元素，仅显示其包含的文本。

2.3.8 dialog——模态对话框

dialog 元素定义对话框或窗口。在默认状态下，dialog 元素处于隐藏状态，可以在 JavaScript 脚本中使用 show()方法显示 dialog 元素，可以使用 close()方法隐藏 dialog 元素。

dialog 元素包含 open 属性，用来设置 dialog 元素打开，用户可与之交互。

【示例 1】下面示例演示了一个打开的对话框，效果如图 2.9 所示。

```
<dialog open>打开的对话框</dialog>
```

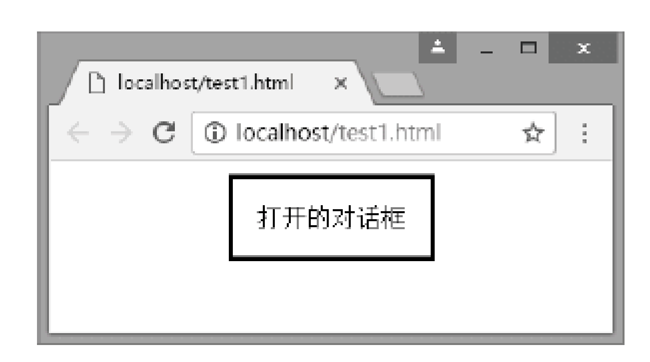
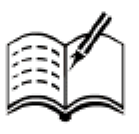


图 2.9 打开的对话框

【示例 2】下面示例演示了如何使用 JavaScript 脚本控制对话框的显示或隐藏。

```
<style>
#dg { width: 60%; text-align: center; }
#dg   label {display:block; margin:6px;}
::backdrop { background-color:black;}
</style>
<input id="btn" type="button" value="打开对话框">
<dialog id="dg">
    <h2>用户登录</h2>
    <main>
        <form>
            <label for="txtName" value="用户名："/>
                <input type="text" id="txtName"/>
            </label>
            <label for="txtPassword" value="密   码："/>
                <input type="password" id="txtPassword"   autofocus/>
            </label>
            <input type="button" value="登 录" />
            <input id="cls" type="button" value="关 闭" />
        </form>
    </main>
</dialog>
<script>
var btn = document.getElementById("btn");
var dg = document.getElementById("dg");
var cls = document.getElementById("cls");
btn.onclick = function(){
    dg.showModal();
    dg.returnValue='对话框的值';
}
```





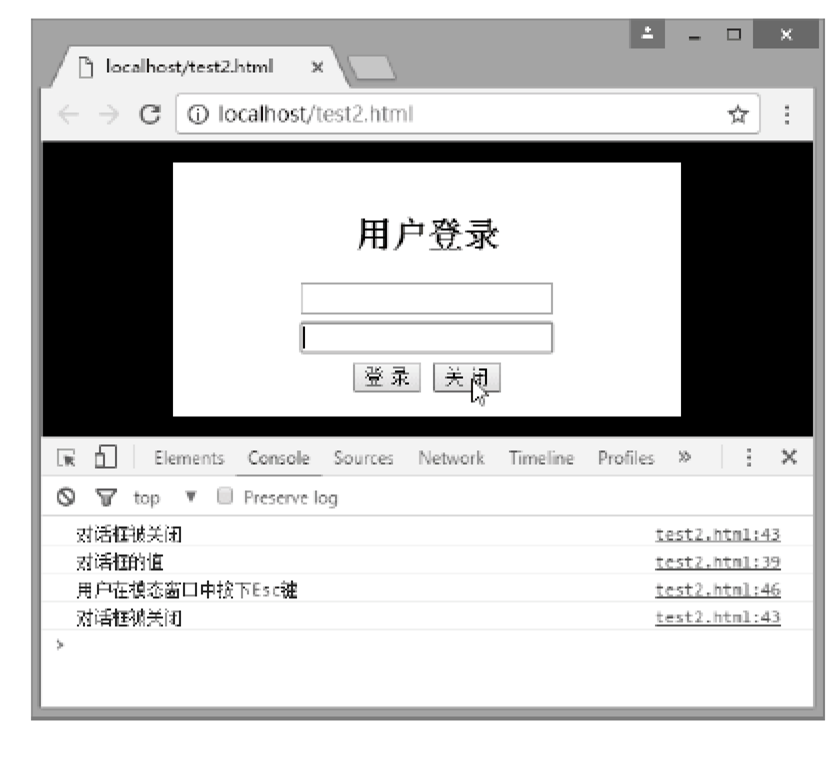
Note

```
    cls.onclick = function(){
        dg.close();
        console.log( dg.returnValue );
    }
    dg.onclose = function(){
        console.log('对话框被关闭');
    }
    dg.oncancel = function(){
        console.log('用户在模态窗口中按下 Esc 键');
    }
</script>
```

在示例页面中，显示一个“打开对话框”按钮，页面打开时 dialog 元素处于隐藏状态，单击“打开对话框”按钮后，dialog 元素变为显示状态。dialog 元素中放置一个“关闭”按钮，单击该按钮后 dialog 元素变为隐藏状态，效果如图 2.10 所示。



（a）默认状态



（b）打开状态

图 2.10　打开对话框

在上面代码中，可以使用 dialog 元素的 showModal()方法以模式对话框的形式显示 dialog 元素；如果要在页面打开时即显示 dialog 元素，可以使用 dialog 元素的 open 属性；可以使用 dialog 元素的 returnValue 属性为对话框设置或返回一个返回值。

提示：目前，Chrome 和 Opera 新版本浏览器对其提供完全支持，Firefox 新版本支持基本功能。

2.3.9　bdi——隔离文本



视 频 讲 解

bdi 元素允许设置一段文本，使其脱离其父元素的文本方向设置。在发布用户评论或其他无法完全控制的内容时，该标签很有用。

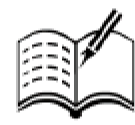
【示例】下面示例将用户名从周围的文本方向设置中隔离出来。

```
<ul>
    <li>用户<bdi>admin</bdi>: 70 分</li>
    <li>用户<bdi>lisi</bdi>: 80 分</li>
    <li>用户<bdi>zhangsan</bdi> : 90 分</li>
</ul>
```

目前，只有 Firefox 和 Chrome 浏览器支持 bdi 元素。



视频讲解



Note

2.3.10 wbr——换行断点

wbr 元素定义在文本中的何处适合添加换行符。如果单词太长，或者担心浏览器会在错误的位置换行，那么可以使用 wbr 元素来添加单词换行点，避免浏览器随意换行。

目前，除了 IE 浏览器外，其他主流浏览器都支持 wbr 元素。

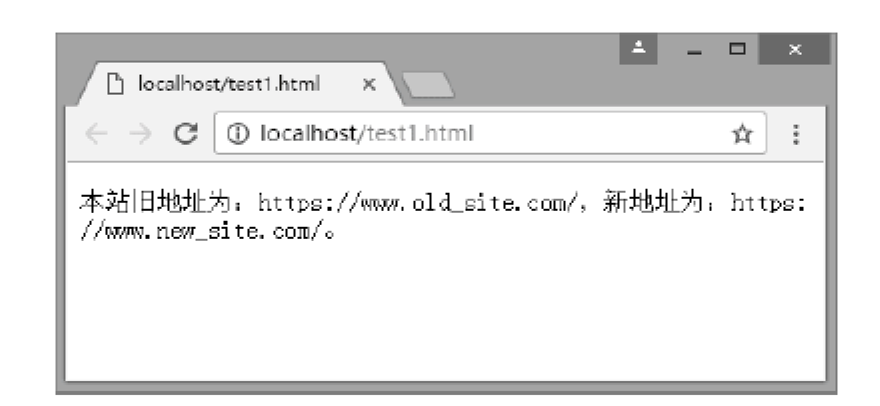
【示例】下面示例为 URL 字符串添加换行符标签，这样当窗口宽度变化时，浏览器会自动根据断点确定换行位置，效果如图 2.11 所示。

```
<p>本站旧地址为：https:<wbr>//<wbr>www.old_site.com/，新地址为：https:<wbr>//<wbr>www.new_site.com/。</p>
```





（a）IE 中换行断点无效



（b）Chrome 中换行断点有效



图 2.11 定义换行断点



视频讲解

2.3.11 ruby、rt、rp——文本注释

ruby 元素可以定义 ruby 注释，如在汉字顶部添加拼音。ruby 元素需要与 rt 或 rp 元素配合使用，其中 rt 和 rp 元素必须位于 ruby 元素内。

☑ rt 元素定义字符（中文注音或字符）的解释或发音。

☑ rp 元素定义备用显示内容，即当浏览器不支持 ruby 元素时的显示内容。

【示例】下面示例演示如何使用 ruby 和 rt 元素为唐诗诗句注音，效果如图 2.12 所示。

```
<style type="text/css">
ruby { font-size: 40px; }
</style>
<ruby>
少<rt>shào</rt>小<rt>xiǎo</rt>离<rt>lí</rt>家<rt>jiā</rt>老<rt>lǎo</rt>大<rt>dà</rt>回<rt>huí</rt>
</ruby>,
<ruby>
乡<rt>xiāng</rt>音<rt>yīn</rt>无<rt>wú</rt>改<rt>gǎi</rt>鬓<rt>bìn</rt>毛<rt>máo</rt>衰<rt>cuī</rt>
</ruby>。
```

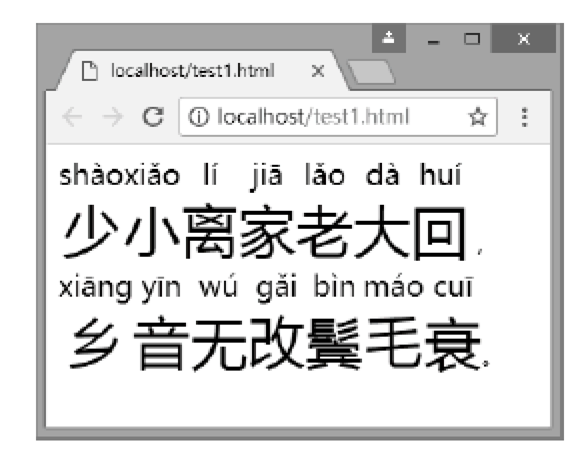
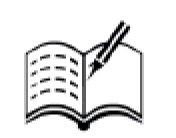


图 2.12 给唐诗注音





视 频 讲 解



Note

2.3.12　command——菜单命令

在 HTML5 中被 HTML4 弃用的 menu 元素被重新定义。使用 menu 元素可以定义命令的列表或菜单，如上下文菜单、工具栏，以及列出表单控件和命令。menu 元素中可以包含 command 和 menuitem 元素，用于定义命令和项目。

【示例 1】下面示例配合使用 menu 和 command 元素，定义一个命令，当单击该命令时，将弹出提示对话框，如图 2.13 所示。

```
<menu>
    <command onclick="alert('Hello World')">命令</command>
</menu>
```

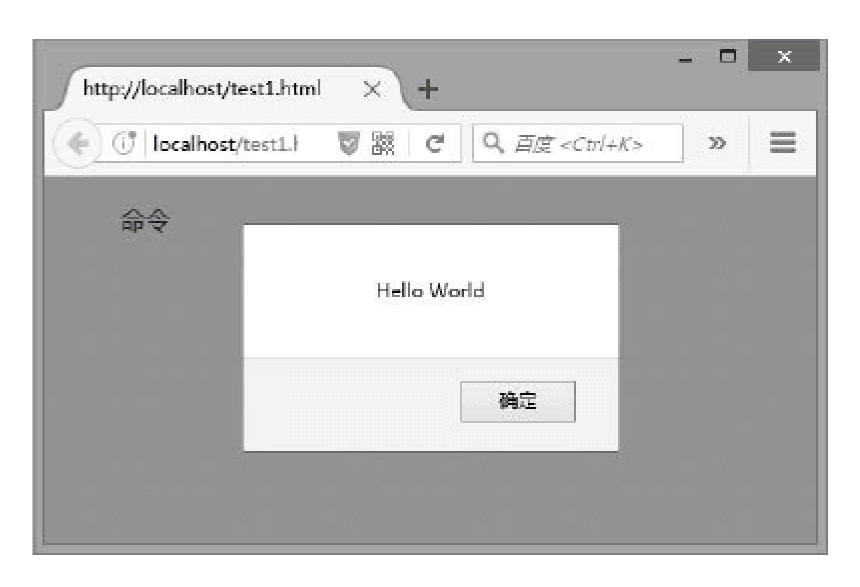


图 2.13　定义菜单命令

command 元素可以定义命令按钮，如单选按钮、复选框或按钮。只有当 command 元素位于 menu 元素内时，该元素才是可见的。否则不会显示这个元素，但是可以用它定义键盘快捷键。

目前，只有 IE 9（更早或更晚的版本都不支持）和 Firefox 支持 command 元素。

command 元素包含很多属性，专门用来定制命令的显示样式和行为，说明如表 2.1 所示。

表 2.1　command 元素属性

属　　性	取　　值	说　　明
checked	checked	定义是否被选中。仅用于 radio 或 checkbox 类型
disabled	disabled	定义 command 是否可用
icon	url	定义作为 command 来显示的图像的 url
label	text	为 command 定义可见的 label
radiogroup	groupname	定义 command 所属的组名。仅在类型为 radio 时使用
type	checkbox、command、radio	定义该 command 的类型。默认值为"command"

【示例 2】下面示例使用 command 元素各种属性定义一组单选按钮命令组，演示效果如图 2.14 所示。目前还没有浏览器完全支持这些属性。

```
<menu>
    <command icon="images/1.png" onclick="alert('男士')" type="radio" radiogroup="group1" label="男士">男士</command>
    <command icon="images/2.png" onclick="alert('女士')" type="radio" radiogroup="group1" label="女士">女士</command>
    <command icon="images/3.png" onclick="alert('未知')" type="radio" radiogroup="group1" label="未知">未
```



```
知</command>
    </menu>
```

menu 元素也包含两个专用属性，简单说明如下：

☑ label：定义菜单的可见标签。

☑ type：定义要显示哪种菜单类型，取值说明如下：

➢ list：默认值，定义列表菜单。一个用户可执行或激活的命令列表（li 元素）。

➢ context：定义上下文菜单。该菜单必须在用户能够与命令进行交互之前被激活。

➢ toolbar：定义工具栏菜单。活动式命令，允许用户立即与命令进行交互。

【示例 3】下面示例使用 type 属性定义了两组工具条按钮，演示效果如图 2.15 所示。

```
    <menu type="toolbar">
        <li>
            <menu label="File" type="toolbar">
                <button type="button" onclick="file_new()">新建...</button>
                <button type="button" onclick="file_open()">打开...</button>
                <button type="button" onclick="file_save()">保存</button>
            </menu>
        </li>
        <li>
            <menu label="Edit" type="toolbar">
                <button type="button" onclick="edit_cut()">剪切</button>
                <button type="button" onclick="edit_copy()">复制</button>
                <button type="button" onclick="edit_paste()">粘贴</button>
            </menu>
        </li>
    </menu>
```

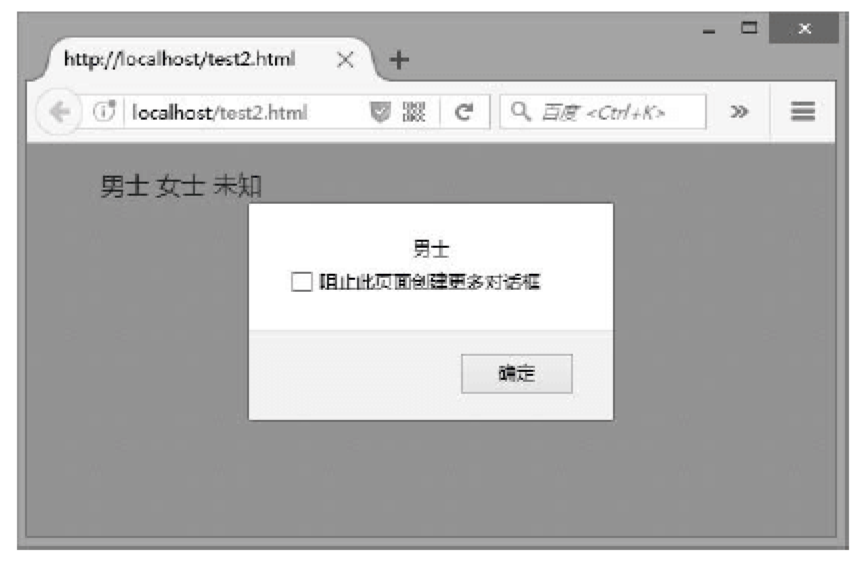



图 2.14　定义单选按钮命令组

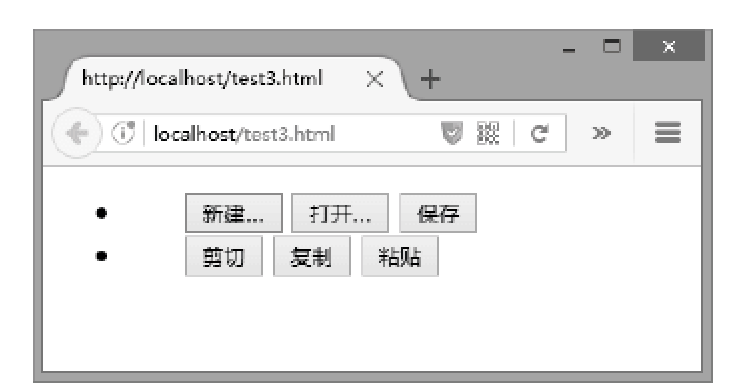


图 2.15　定义工具条命令组

2.4　完善旧元素

HTML5 对 HTML4 部分元素进行了优化，具体说明如下。

2.4.1　a——超链接

HTML5 为 a 元素新增了 3 个属性，简单说明如下：

☑ download：设置被下载的超链接目标。

Note


视频讲解



☑　media：设置被链接文档是被何种媒介/设备优化的。

☑　type：设置被链接文档的 MIME 类型。

【示例】下面示例使用 download 属性设计图片被单击后，直接下载，而不是在新窗口中显示，效果如图 2.16 所示。

```
<a href="images/1.jpg" download="images/1.jpg"><img src="images/1.jpg"/></a>
```



Note

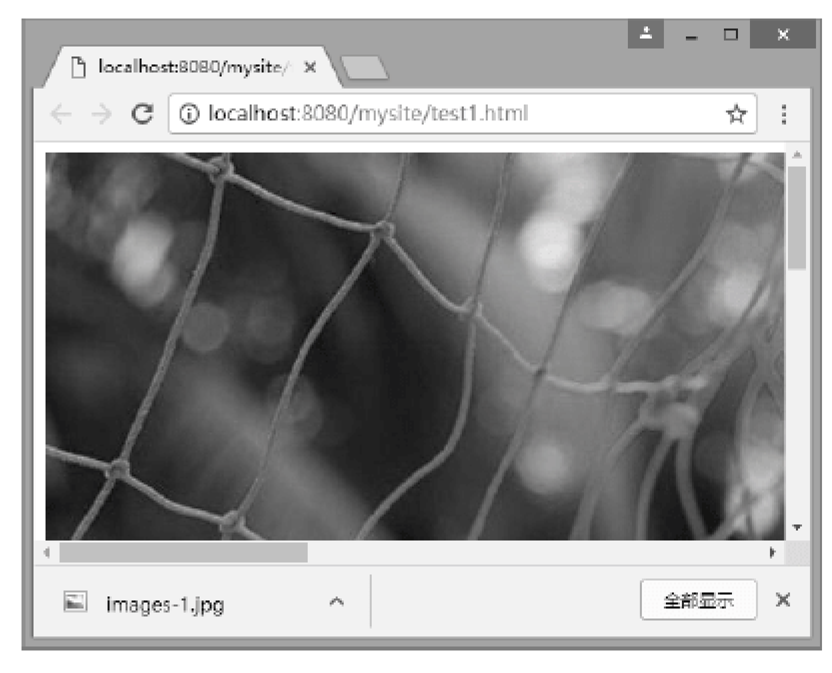


图 2.16　单击下载图片

提示：目前，Chrome、Opera 和 Firefox 版本的浏览器均支持该属性，IE 暂不支持。

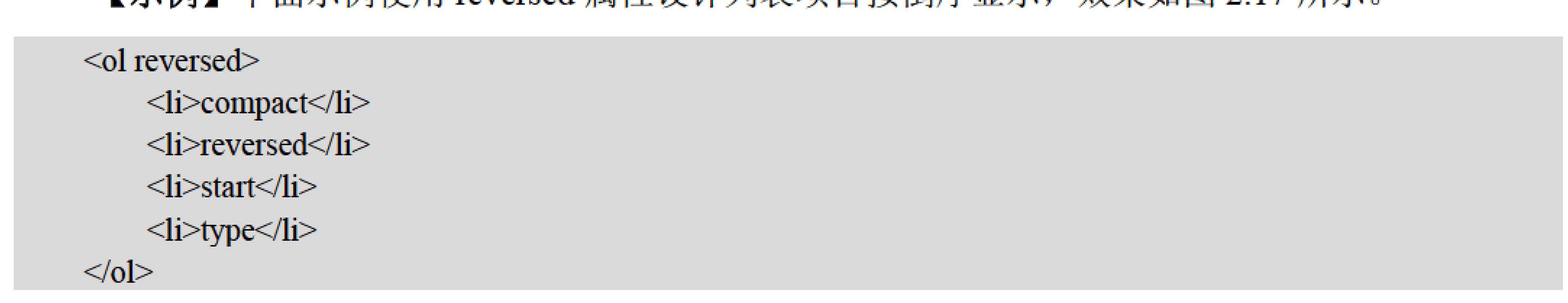
2.4.2　ol——有序列表



视频讲解

HTML5 为 ol 元素新增了 reversed 属性，用来设置列表顺序为降序显示。

【示例】下面示例使用 reversed 属性设计列表项目按倒序显示，效果如图 2.17 所示。



```
<ol reversed>
    <li>compact</li>
    <li>reversed</li>
    <li>start</li>
    <li>type</li>
</ol>
```

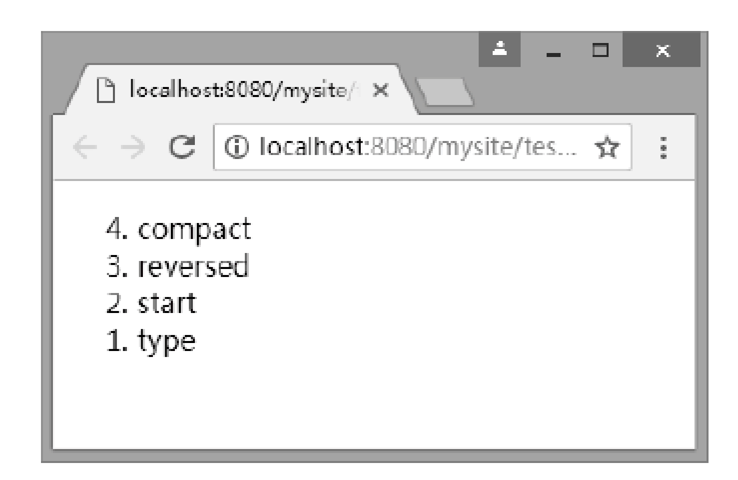


图 2.17　项目列表倒序显示

提示：目前，最新版本的 Chrome、Opera、Firefox 浏览器均支持该属性，IE 暂不支持。

2.4.3　dl——定义列表

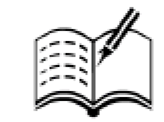


视频讲解

HTML5 重新定义了 dl 元素，允许 dl 列表包含多个带名字的列表项。每一项包含一条或多条带名



字的 dt 元素，用来表示术语，dt 元素后面紧跟一个或多个 dd 元素，用来表示定义。在一个元素内，不允许有相同名字的 dt 元素，即不允许有重复的术语。



Note

【示例】下面示例演示了如何使用 dl 元素对诗句进行逐句解析，效果如图 2.18 所示。

```
<h3>《静夜思》赏析</h3>
<dl>
    <dt><dfn>床前明月光，疑是地上霜。</dfn></dt>
    <dd>诗的前两句，是写诗人在作客他乡的特定环境中一刹那间所产生的错觉。</dd>
    <dt><dfn>举头望明月，低头思故乡。</dfn></dt>
    <dd>诗的后两句，则是通过动作神态的刻画，深化思乡之情。</dd>
</dl>
```



图 2.18　定义列表项目的应用



视频讲解

2.4.4　cite——引用文本

cite 元素表示引用参考，如书籍或者杂志的标题。按照惯例，引用的文本将以斜体显示。一般应把引用文本包裹在 a 元素中，方便用户快速跳转到原出处。

cite 元素还有一个隐藏的功能：可以从文档中自动摘录参考书目。浏览器能够自动整理引用表格，并把它们作为脚注或者独立的文档来显示。

cite 元素的语义已经超过了改变它所包含的文本外观的作用，它使浏览器能够以各种实用的方式来向用户表达文档的内容。

【示例】下面示例简单演示了 cite 元素的应用，效果如图 2.19 所示。

```
<p>宋词是一种相对于古体诗的新体诗歌之一，标志宋代文学的最高成就。宋词句子有长有短，便于歌唱。因是合乐的歌词，故又称曲子词、乐府、乐章、长短句、诗余、琴趣等。</p>
<p> 来 自 百 度 百 科 <a  href="https://baike.baidu.com/item/%E5%AE%8B%E8%AF%8D/365879?fr=aladdin" target="_blank"><cite>宋词</cite></a></p>
```

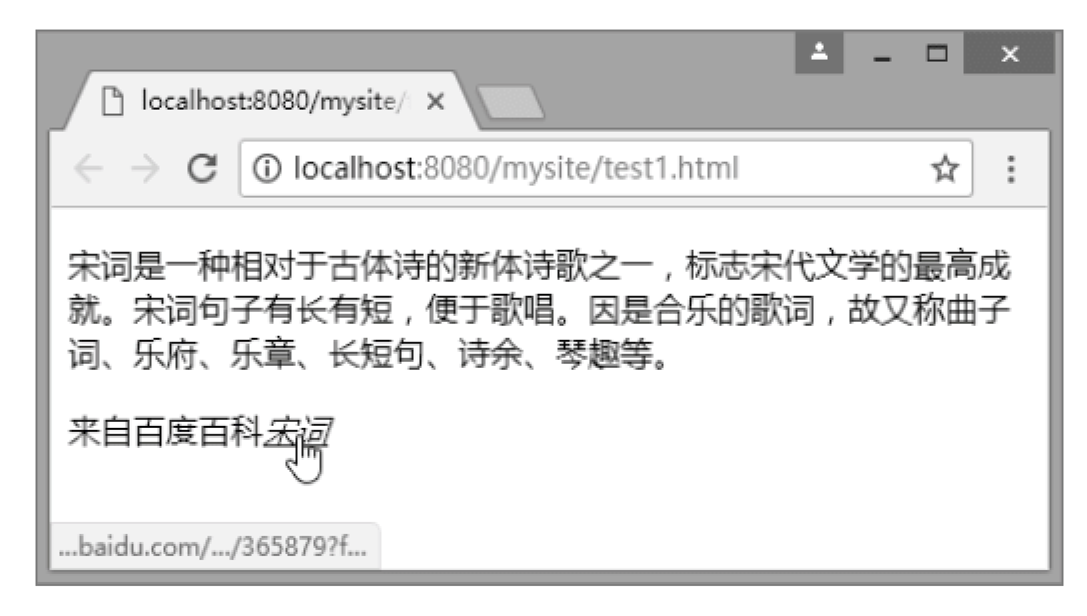
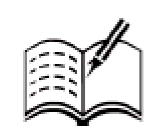


图 2.19　使用 cite 元素





2.4.5　small——小号字体

small 元素本来用来定义小号字体效果，HTML5 对其进行了重新定义，使其由原来的通用展示性元素变为更具体的、专门用来标识所谓“小字印刷体”的元素，通常用在诸如免责声明、注意事项、法律规定、与版权相关等法律性声明文字中，同时不允许应用在页面主内容中，只允许被当作辅助信息，以 inline 方式内嵌在页面上。

同时，small 元素也不意味着元素中内容字体会变小，要将字体变小，需要使用 CSS 样式表。

2.4.6　iframe——浮动框架

HTML5 主要从安全性方面增强 iframe 元素，新增了 3 个属性，简单说明如下：

☑　sandbox：启用一系列对 iframe 中内容的额外限制，取值包括：""、allow-forms（允许表单提交）、allow-same-origin（允许同源访问）、allow-scripts（允许执行脚本）、allow-top-navigation（允许框架访问）。

☑　seamless：定义 iframe 看上去像是包含文档的一部分，取值为 seamless（无缝嵌入），或者不设置。

☑　srcdoc：规定在 iframe 中显示的 HTML 内容，取值为 HTML 代码。

HTML5 为 iframe 元素增加 sandbox 属性，是出于安全性方面的原因，对 iframe 元素内的内容是否允许显示，表单是否允许被提交，以及脚本是否允许被执行等方面进行一些限制。

通过设置 iframe 元素的 sandbox 属性，iframe 元素内显示的页面被添加如下所示的限制。

☑　该页面中的插件被禁用。

☑　该页面中的表单被禁止提交。

☑　该页面中的 JavaScript 脚本代码被禁止运行。

☑　如果单击该页面内的超链接，将把浏览器窗口或 iframe 元素之外的任何内容导航到 iframe，则该超链接被禁用。

☑　该页面被视为来自一个单独的源，所以禁止加载该页面中来自服务器端的内容，禁止该页面与服务器端进行交互，同时禁止加载页面中从 Cookie 或 Web Storage 中读出的内容。

提示：sandbox 属性允许指定多个属性值，属性值与属性值中间用空格分隔。

2.4.7　script——脚本



视频讲解

HTML5 为 script 元素新增 async 属性，规定异步执行脚本，仅适用于外部脚本，取值为 async。

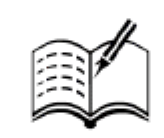
【示例 1】下面示例演示了 async 属性的应用。

```
<script src="test1.js" async onload="ok()"></script>
<script>
console.log("内部脚本");
</script>
```

设计在页面中导入外部脚本文件 test1.js，该文件的代码如下：

```
function ok(){
    console.log("外部脚本");
}
```





在 Chrome 浏览器中预览，可以看到页面内部脚本先被执行，最后才执行异步导入的脚本文件代码，效果如图 2.20 所示。

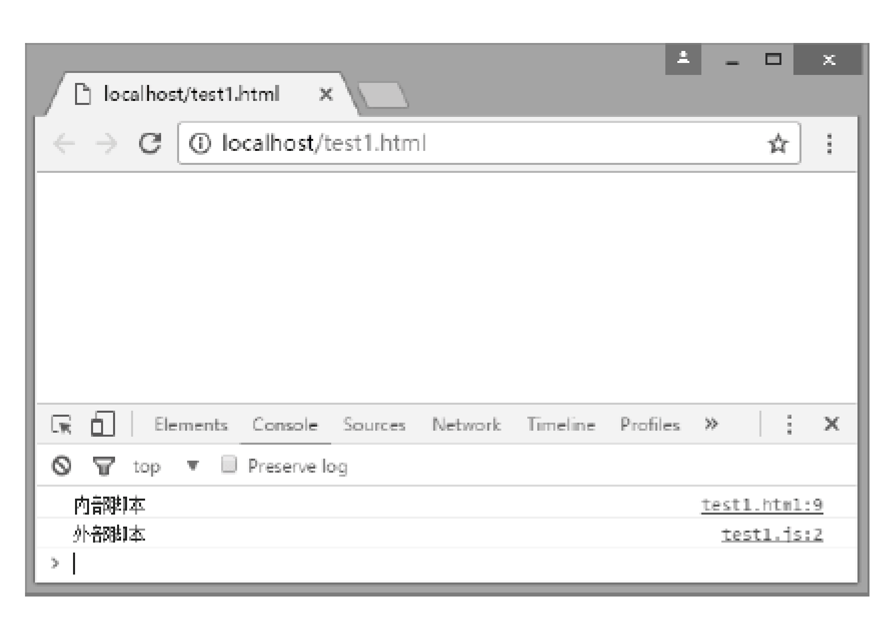


图 2.20　异步加载 JavaScript 脚本

【示例 2】如果在 script 元素中删除 async 属性，则可以看到先等到外部 JavaScript 脚本文件加载完毕之后，才执行内部脚本，效果如图 2.21 所示。

```
<script src="test1.js" onload="ok()"></script>
<script>
console.log("内部脚本");
</script>
```

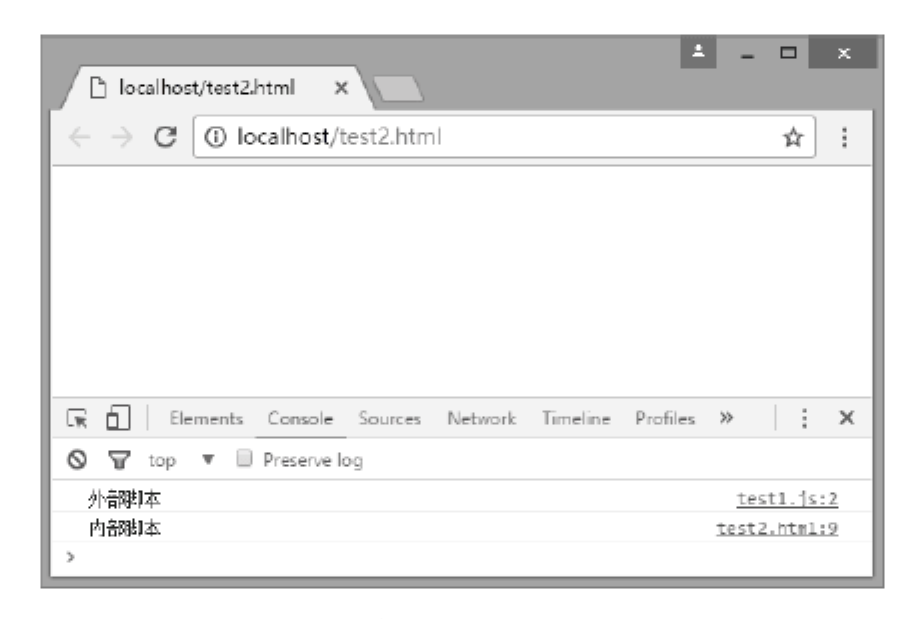


图 2.21　同步加载 JavaScript 脚本

2.5　HTML5 新的全局属性

HTML5 除了支持 HTML4 原有的全局属性之外，还添加了 8 个新的全局属性。所谓全局属性是指可以用于任何 HTML 元素的属性。

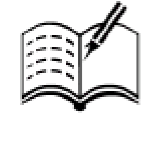
2.5.1　contentEditable——可编辑内容


视频讲解

contentEditable 属性的主要功能是允许用户在线编辑元素中的内容。contentEditable 是一个布尔值属性，可以被指定为 true 或 false。

注意，该属性还有个隐藏的 inherit（继承）状态，属性为 true 时，元素被指定为允许编辑；属性为 false 时，元素被指定为不允许编辑；未指定 true 或 false 时，则由 inherit 状态来决定，如果元素的父元素是可编辑的，则该元素就是可编辑的。

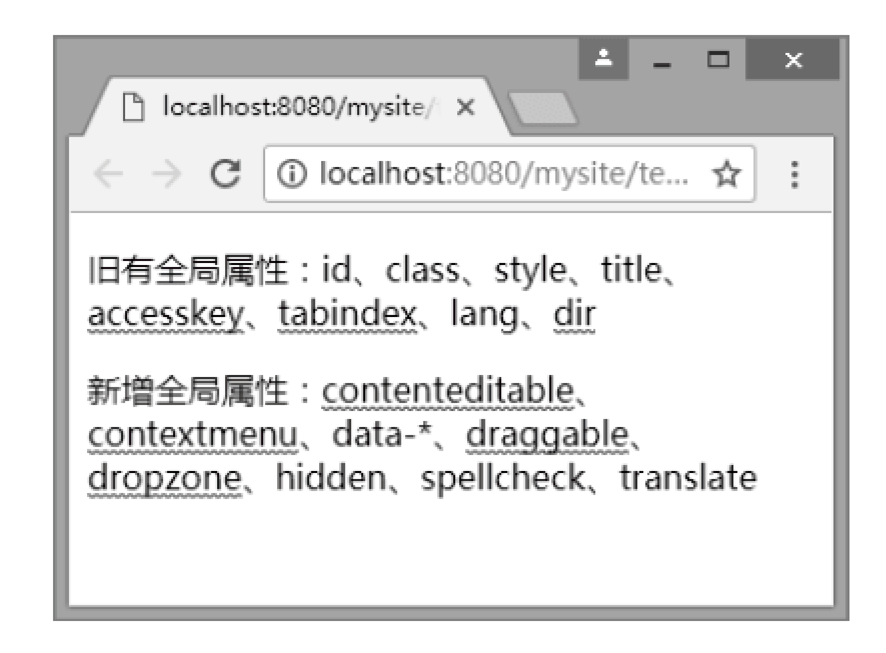




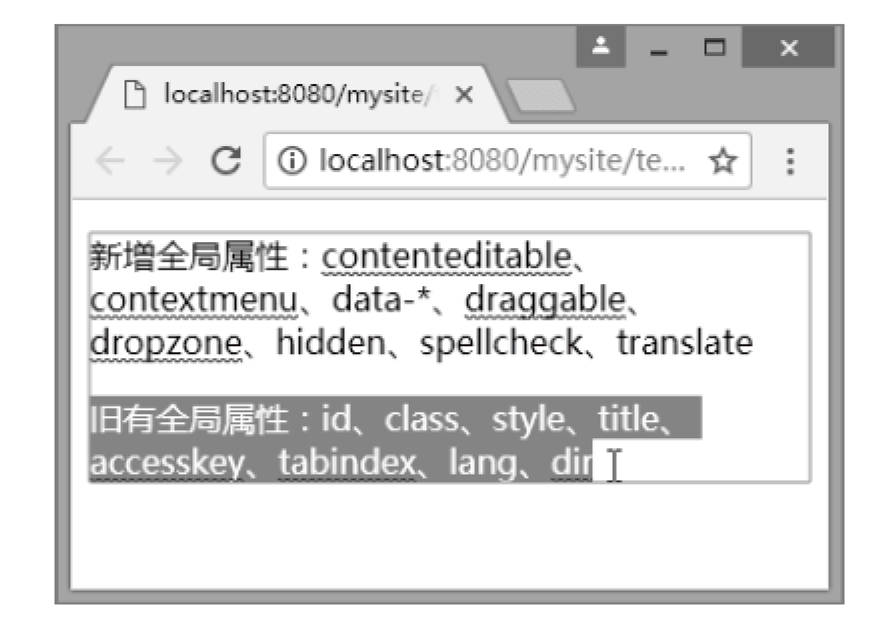
Note

【示例】在下面示例中为正文文本包含框<div>标签加上 contentEditable 属性后，该包含框包含的文本就变成可编辑的了，浏览者可自行在浏览器中修改内容，执行结果如图 2.22 所示。

```
<div contentEditable="true">
    <p>旧有全局属性：id、class、style、title、accesskey、tabindex、lang、dir</p>
    <p>新增全局属性：contenteditable、contextmenu、data-*、draggable、dropzone、hidden、spellcheck、translate</p>
</div>
```



（a）原始列表



（b）编辑列表项项目

图 2.22　可编辑文本

在编辑完元素中的内容后，如果想要保存其中内容，只能使用 JavaScript 脚本把该元素的 innerHTML 发送到服务器端进行保存，因为改变元素内容后该元素的 innerHTML 内容也会随之改变，目前还没有特别的 API 来保存编辑后元素中的内容。

提示： 在 JavaScript 脚本中，元素还具有一个 isContentEditable 属性，当元素可编辑时，该属性值为 true；当元素不可编辑时，该属性值为 false。利用这个属性，可以实现对编辑数据的后期操作。

2.5.2　contextmenu——快捷菜单



视 频 讲 解

contextmenu 属性用于定义元素的上下文菜单。所谓上下文菜单，就是会在用户右击元素时出现。

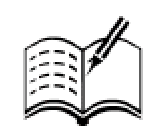
【示例】下面示例使用 contextmenu 属性定义<div>元素的上下文菜单，其中 contextmenu 属性的值是要打开的<menu>元素的 id 属性值。

```
<div contextmenu="mymenu">上下文菜单
    <menu type="context" id="mymenu">
        <menuitem label="微信分享"></menuitem>
        <menuitem label="微博分享"></menuitem>
    </menu>
</div>
```

当用户右击元素时，会弹出一个上下文菜单，从中可以选择指定的快捷菜单项目，如图 2.23 所示。

提示： 目前只有 Firefox 支持 contextmenu 属性。





Note

图 2.23　打开上下文菜单



视频讲解

2.5.3　data——自定义属性

使用 data-*属性可以自定义用户数据。具体应用包括：

☑　data-*属性用于存储页面或元素的私有数据。

☑　data-*属性赋予所有 HTML 元素嵌入自定义属性的能力。

存储的自定义数据能够被页面的 JavaScript 脚本利用，以创建更好的用户体验，方便 Ajax 调用或服务器端数据库查询。

data-*属性包括两部分：

☑　属性名：不应该包含任何大写字母，并且在前缀 "data-" 之后必须有至少一个字符。

☑　属性值：可以是任意字符串。

当浏览器解析时，会忽略前缀"data-"，取用其后的自定义属性。

【示例 1】下面示例使用 data-*属性为每个列表项目定义一个自定义属性 type。这样在 JavaScript 脚本中可以判断每个列表项目包含信息的类型。

```
<ul>
    <li data-animal-type="bird">猫头鹰</li>
    <li data-animal-type="fish">鲤鱼</li>
    <li data-animal-type="spider">蜘蛛</li>
</ul>
```

【示例 2】以上面示例为基础，下面示例使用 JavaScript 脚本访问每个列表项目的 type 属性值，演示效果如图 2.24 所示。

```
<ul>
    <li data-animal-type="bird">猫头鹰</li>
    <li data-animal-type="fish">鲤鱼</li>
    <li data-animal-type="spider">蜘蛛</li>
</ul>
<script>
var lis = document.getElementsByTagName("li");
for(var i=0; i<lis.length; i++){
    console.log(lis[i].dataset.animalType);
}
</script>
```



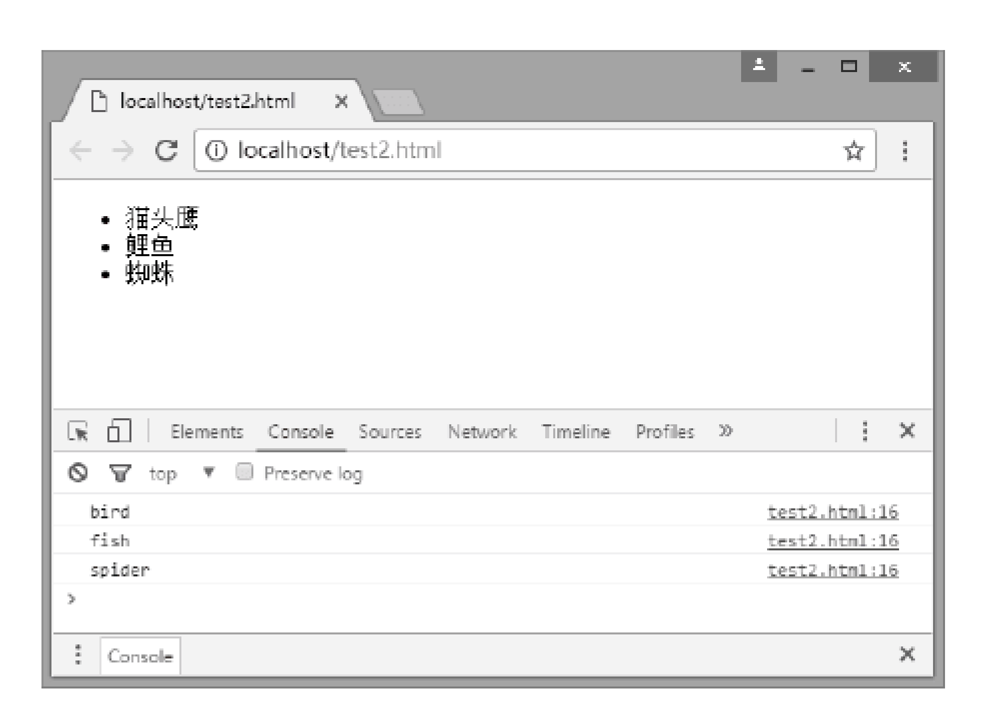
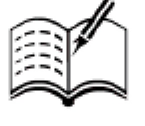


图 2.24　访问列表项目的 type 属性值



访问元素的自定义属性，可以通过元素的 dataset.对象获取，该对象存储了元素所有自定义属性的值。访问规则与 CSS 脚本化访问相同。对于复合属性名，通过驼峰命名法访问，如 animal-type，访问时使用 animalType，避免连字符在脚本中引发的歧义。

注意：目前 IE 暂不支持这种访问方式。

2.5.4　draggable——可拖动

draggable 属性可以定义元素是否可以被拖动。属性取值说明如下：

☑　true：定义元素可拖动。

☑　false：定义元素不可拖动。

☑　auto：定义使用浏览器的默认行为。

draggable 属性常用在拖放操作中，详细说明请参考第 15 章拖放 API。

2.5.5　dropzone——拖动数据

dropzone 属性定义在元素上拖动数据时，是否复制、移动或链接被拖动数据。属性取值说明如下：

☑　copy：拖动数据会产生被拖动数据的副本。

☑　move：拖动数据会导致被拖动数据被移动到新位置。

☑　link：拖动数据会产生指向原始数据的链接。

例如：

```
<div dropzone="copy"></div>
```

提示：目前所有主流浏览器都不支持 dropzone 属性。

2.5.6　hidden——隐藏



视 频 讲 解

在 HTML5 中，所有元素都包含一个 hidden 属性。该属性设置元素的可见状态，取值为一个布尔值，当设为 true 时，元素处于不可见状态；当设为 false 时，元素处于可见状态。

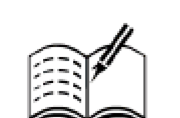
【示例】下面使用 hidden 属性定义段落文本隐藏显示。

```
<p hidden><img src="images/1.jpg" width="200" /></p>
```

hidden 属性可用于防止用户查看元素，直到匹配某些条件，如选择了某个复选框。然后，在页



面加载之后，可以使用 JavaScript 脚本删除该属性，删除之后该元素变为可见状态，同时元素中的内容也即时显示出来。





Note

视频讲解

提示：除了 IE，所有主流浏览器都支持 hidden 属性。

2.5.7 spellcheck——语法检查

spellcheck 属性定义是否对元素进行拼写和语法检查。可以对以下内容进行拼写检查：

☑ input 元素中的文本值（非密码）。

☑ <textarea>元素中的文本。

☑ 可编辑元素中的文本。

spellcheck 属性是一个布尔值的属性，取值包括 true 和 false，为 true 时表示对元素进行拼写和语法检查，为 false 时则不检查元素。用法如下所示：

```
<!--以下两种书写方法正确-->
<textarea spellcheck="true" >
<input type=text spellcheck=false>
<!--以下书写方法为错误-->
<textarea spellcheck >
```

注意，如果元素的 readOnly 属性或 disabled 属性设为 true，则不执行拼写检查。

【示例】下面示例设计两段文本，第一段文本可编辑、可语法检查；第二段文本可编辑，但不允许语法检查。当编辑文本时，第一段文本显示检查状态，而第二段忽略，如图 2.25 所示。

```
<div contentEditable="true">
    <p spellcheck="true">旧有全局属性：id、class、style、title、accesskey、tabindex、lang、dir</p>
    <p spellcheck="false">新增全局属性：contenteditable、contextmenu、data-*、draggable、dropzone、hidden、spellcheck、translate</p>
</div>
```

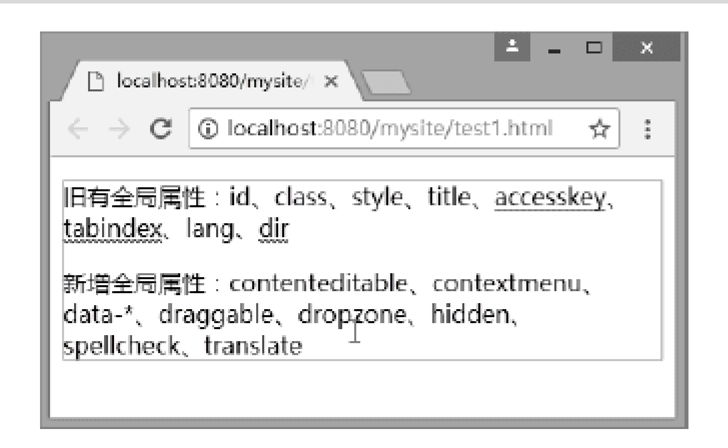


图 2.25　段落文本检查状态比较

2.5.8 translate——可翻译

translate 属性定义是否应该翻译元素内容。取值说明如下：

☑ yes：定义应该翻译元素内容。

☑ no：定义不应翻译元素内容。

【示例】下面示例演示了如何使用 translate 属性。

```
<p translate="no">请勿翻译本段。</p>
<p>本段可被译为任意语言。</p>
```



提示：目前，所有主流浏览器都无法正确地支持 translate 属性。



2.6　HTML5 文档大纲

HTML5 对如何处理位于 article、aside、nav 和 section 等结构元素中的 h1～h6 有一套算法。该算法通常称为 HTML5 文档大纲，文档大纲不会对页面结构造成破坏，也不会影响布局。不过，目前还没有浏览器实现这套算法，在屏幕阅读器中只有 JAWS（一款运行于 Windows 下的屏幕阅读器）支持，而它的实现还存在问题。鉴于此，W3C 已经将文档大纲列入可能从最终定稿的规范中移除的特性。

Note

2.6.1　定义文档节段

在 HTML5 文档中，节段是一个很重要的概念，它表示一个语义独立的内容块。能够定义节段的元素包括：body、section、article、aside、nav、header、footer。



线上阅读



视频讲解

节段可以相互嵌套，形成嵌套的结构层次关系。每个节段都必须有自己的标题，即使是嵌套的节段，也必须有自己的标题，标题可以使用 h1、h2、h3、h4、h5、h6 元素之一标识。

详细示例演示说明，请扫码学习。

2.6.2　隐式分节

HTML5 分节元素不会强制性定义大纲，为了与 HTML4 保持兼容，有一种方式来定义节段，而不需要分节元素，这种方式就是隐式分节。



线上阅读



视频讲解

当标题元素（h1～h6）不是父节段的第一个标题时，它会隐式定义一个新的节段。这种隐式节段通过在父节点中与之前标题的相对级别来确定。如果比之前的标题级别更低，那么就会定义一个新的子节段。

详细示例演示说明，请扫码学习。

2.6.3　特殊分节

1. 分节根

分节根拥有独立的大纲体系，其内的节段与外部大纲没有联系。使用 blockquote、details、fieldset、figure、td 可以定义分节根元素。



线上阅读



视频讲解

2. 主纲之外的节段

下面 4 个元素用来定义不属于文档主要大纲中的节段：aside、nav、header、footer。

详细示例演示说明，请扫码学习。

2.7　案 例 实 战



视频讲解

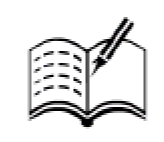
HTML5 定义了一组新的结构化语义标记，使用它们来描述网页内容，确保 HTML5 文档结构的简洁、友好。虽然也可以使用 HTML4 的 div 和 span 进行代替设计，但是 HTML5 新元素简化了页面



设计，明确的语义结构更适合搜索引擎检索和抓取。本节将借助 HTML5 新元素设计一个博客首页。

【操作步骤】

第 1 步，新建 HTML5 文档，保存为 test1.html。



Note

第 2 步，根据上面各节介绍的知识，开始构建个人博客首页的框架结构。在设计结构时，最大限度地选用 HTML5 新结构元素，所设计的模板页面基本结构如下所示。

```
<header>
    <h1>[网页标题]</h1>
    <h2>[次级标题]</h2>
    <h4>[标题提示]</h4>
</header>
<main>
    <nav>
        <h3>[导航栏]</h3>
        <a href="#">链接 1</a> <a href="#">链接 2</a> <a href="#">链接 3</a>
    </nav>
    <section>
         <h2>[文章块]</h2>
        <article>
            <header>
                <h1>[文章标题]</h1>
            </header>
            <p>[文章内容]</p>
            <footer>
                <h2>[文章脚注]</h2>
            </footer>
        </article>
    </section>
    <aside>
        <h3>[辅助信息]</h3>
    </aside>
    <footer>
        <h2>[网页脚注]</h2>
    </footer>
</main>
```

整个页面包括两部分：标题部分和主要内容部分。标题部分又包括：网站标题、副标题和提示性标题信息；主要内容部分包括：导航、文章块、侧边栏、脚注。文章块包括 3 部分：标题部分、正文部分和脚注部分。

第 3 步，使用 HTML5 大纲工具来检查模板页面的结构设计是否合理。访问 http://gsnedders.html5.org/outliner/ 页面，在该页面提交本地的模板文档 test1.html，则可以看到如图 2.26 所示的大纲信息。

检查结果说明本示例模板页面设计：结构合理。

第 4 步，在模板页面基础上，开始细化本示例博客首页。下面仅给出本例首页的静态页面结构，如果用户需要后台动态生成内容，则可以考虑在模板结构基础上另外设计。把 test1.html 另存为 test2.html，细化后的静态首页效果如图 2.27 所示。

 **提示：**限于篇幅，本节没有展示完整的页面代码，读者可以通过本节示例源代码了解完整的页面结构。







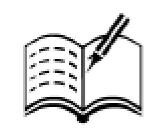


图 2.26　检查大纲结构是否合理



图 2.27　细化后的首页页面效果

第 5 步，设计页面样式部分代码。这里主要使用了 CSS3 的一些新特性，如圆角（border-radius）和旋转变换等，通过 CSS 设计的页面显示效果如图 2.28 所示。相关 CSS3 技术介绍请参阅下面章节内容。

考虑到本章重点学习 HTML5 新元素的应用，所以本节示例不再深入讲解 CSS 样式代码的设计过程，感兴趣的读者可以参考本节示例源代码中的 test3.html 文档。

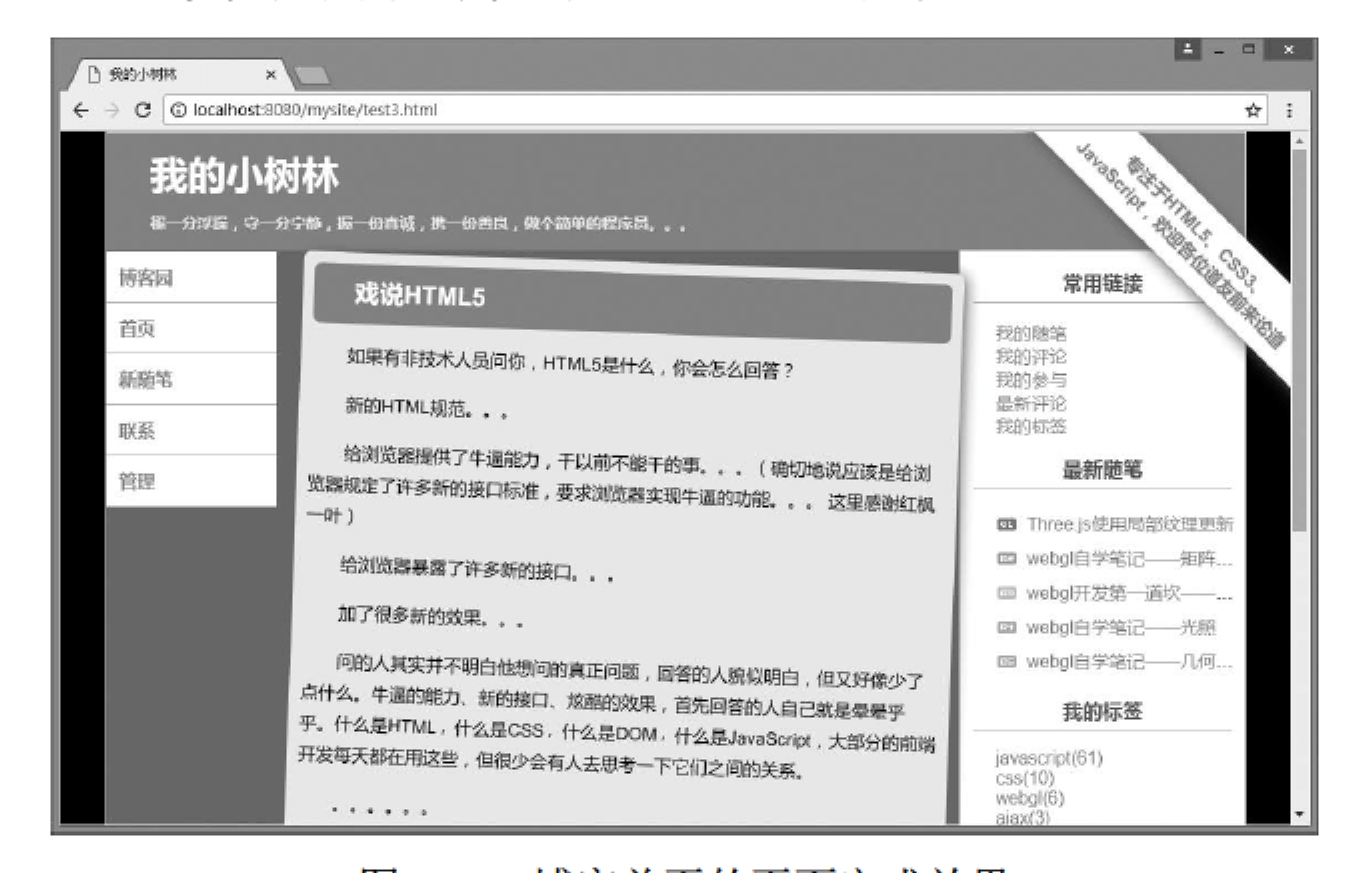
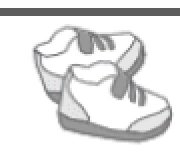
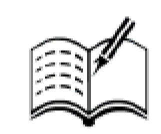


图 2.28　博客首页的页面完成效果



示 例 效 果





Note

第 6 步，对于早期版本浏览器，或者不支持 HTML5 的浏览器，需要添加一个 CSS 样式，因为未知元素默认为行内显示（display:inline），对于 HTML5 结构元素来说，我们需要让它们默认为块状显示。

```
article, section, nav, aside, main, header, hgroup, footer {
    display: block;
}
```

第 7 步，一些浏览器不允许样式化不支持的元素。这种情形出现在 IE8 及以前的浏览器中，因此还需要使用下面 JavaScript 脚本进行兼容。

```
<!--[if lt IE 9]>
  <script>
    document.createElement("article");
    document.createElement("section");
    document.createElement("nav");
    document.createElement("aside");
    document.createElement("main");
    document.createElement("header");
      document.createElement("hgroup");
    document.createElement("footer");
  </script>
<![endif]-->
```

第 8 步，如果浏览器禁用了脚本，则不会显示，可能会出问题。因为这些元素定义整个页面的结构。为了预防这种情况，可以加上<noscript>标签进行提示。

```
<noscript>
   <h1>警告</h1>
   <p>因为你的浏览器不支持 HTML5，一些元素是模拟使用 JavaScript。不幸的是，您的浏览器已禁用脚本。请启用它以显示此页。</p>
</noscript>
```

2.8 在线练习

使用 HTML 结构标签设计各种网页模块。





HTML5 表单

HTML5 Web Forms 2.0 对 HTML4 表单的功能进行全面升级，它在保持了简便易用的特性的同时，增加了许多内置的控件或者控件属性来满足用户的需求，同时减少了开发人员的编程。本章将详细介绍 HTML5 新增的表单类型和属性。

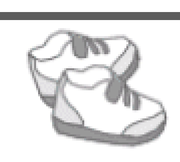
权威参考：http://www.w3.org/Submission/web-forms2/

浏览器支持：https://caniuse.com/、www.wufoo.com/html5。前者上的信息通常比后者上的信息更新一些，不过后者仍然是有关 HTML5 表单信息的一个重要资源。

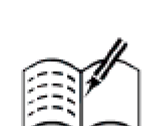
权威参考

【学习重点】

- 使用不同类型的文本框。
- 正确设置新属性。
- 熟悉表单元素和属性。
- 灵活使用 HTML5 新功能设计表单页面。



3.1 HTML5 表单特性



Note

HTML5 表单新增了很多功能，我们将在本章各节中详细说明。下面简单列举几个对于开发者来说具有重要价值的特性。提示：浏览器对最后 3 个特性的支持不是很好，W3C 可能会在最终的 HTML5 版本中放弃规范。

1．新的控件类型

HTML5 新增一系列新的控件，这些表单控件具备类型检查的功能，如 URL 输入框、Email 输入框等。

```
<input type="url" />
<input type="email" />
```

2．日期选择器和颜色选择器

在 HTML5 之前，用户一般使用 JavaScript 和 CSS 设计日期选择器和颜色选择器，这样费时费力，且使用不是很友好。简便的方法就是借助相关框架，如 Dojo、YUI 等类库。

```
<input type="date" />
<input type="color" />
```

3．改进文件上传控件

在 HTML5 中，文件上传控件变得非常强大和易用，用户可以使用一个控件上传多个文件，自行规定上传文件的类型（accept），甚至可以设定每个文件最大的大小（maxlength）。

4．内建表单校验系统

HTML5 为不同类型的输入控件各自提供了新的属性，来控制这些控件的输入行为，如必填项 required 属性，为数字类型控件提供的 max、min 等。在提交表单时，一旦校验错误，浏览器将不执行提交操作，而会显示相应的检验错误信息。

```
<input type="text" required />
<input type="number" min=10   max=100 />
```

5．XML Submission

form 的编码格式一般为 application/x-www-form-urlencoded。这种格式的数据传送到服务器端，可以方便地存取。HTML5 将提供一种新的数据格式——XML Submission，即 application/x-www-form+xml，这样服务器端将直接接收到 XML 形式的表单数据。

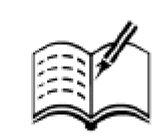
```
<submission>
    <field name="name" index="0">Peter</field>
    <field name="password" index="0">password</field>
</submission>
```

6．外联数据源

在 HTML5 之前，为 select 下拉列表动态添加很多选项，非常烦琐，而这些选项多来自于数据库，如分类列表、商品列表等。HTML5 支持 data 属性，为 select 控件提供外联数据源。

```
<select data="http://domain/options"></select>
```





Note

7．重复（repeat）的模型

HTML5 提供一套重复机制来帮助用户构建一些重复输入列表，如 add、remove、move-up、move-down 的按钮类型，通过这一套重复的机制，开发人员可以非常方便地实现经常用到编辑列表，这是一个很常规的模式，我们可以增加一个条目、删除某个条目、移动某个条目等。

3.2 新的 Input 类型

HTML5 新增了多个输入型表单控件，通过使用这些新增的表单输入类型，可以实现更好的输入控制和验证。目前，Opera 浏览器支持最好，但在所有主流浏览器中都可以使用，即使不被支持，仍然可以显示为普通的文本框。



视频讲解

3.2.1 email——Email 地址框

email 类型的 input 元素是一种专门用于输入 Email 地址的文本框，在提交表单的时候，会自动验证 Email 输入框的值。如果不是一个有效的电子邮件地址，则该输入框不允许提交该表单。

【示例】下面是 email 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入您的 Email 地址：<input type="email" name="user_email" /><br />
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.1 所示。如果输入了错误的 Email 地址格式，单击“提交”按钮时会出现如图 3.2 所示的提示。

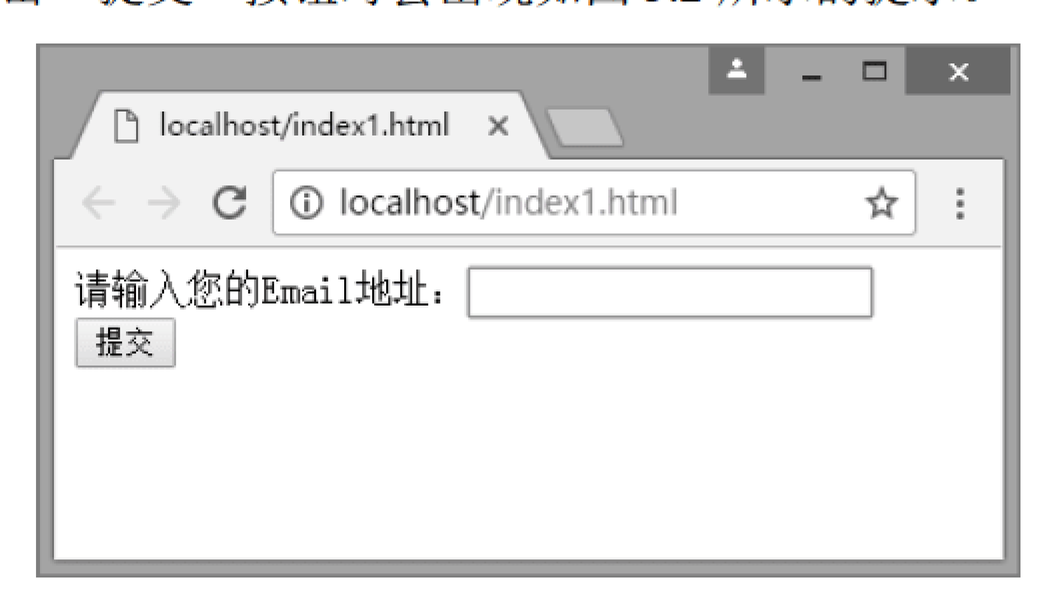




图 3.1 email 类型的 input 元素示例

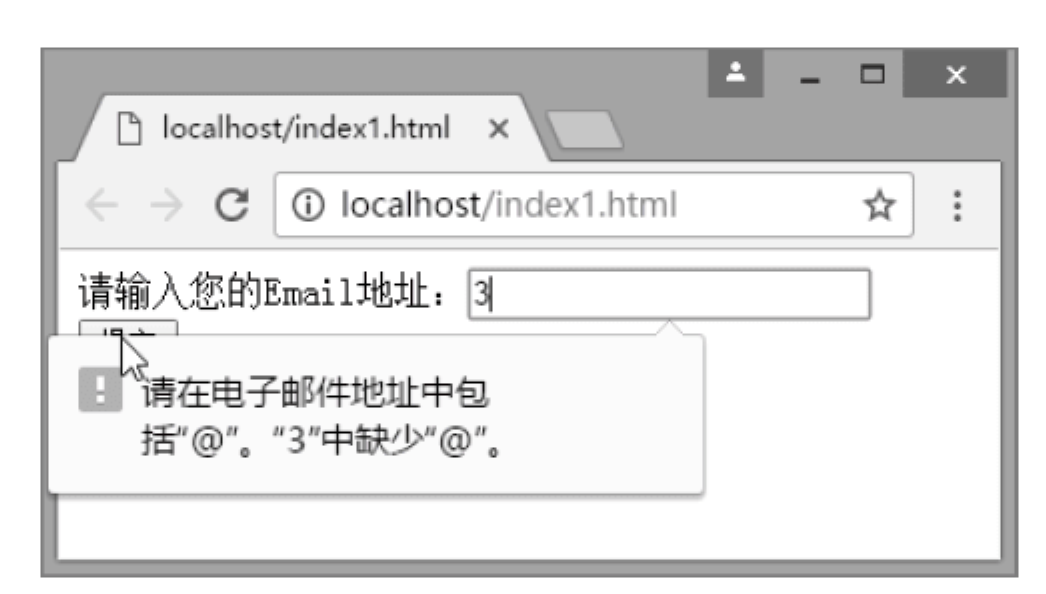
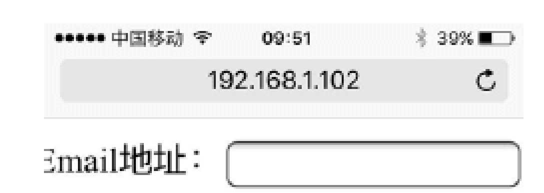


图 3.2 检测到不是有效的 Email 地址

其中 demo_form.php 表示提交给服务器端的处理文件。对于不支持 type="email"的浏览器来说，将会以 type="text"来处理，所以并不妨碍旧版浏览器浏览采用 HTML5 中 type="email"输入框的网页。

如果将 email 类型的 input 元素用在手机浏览器中，则会更加凸显其优势。例如，如果使用 iPhone 或 iPod 中的 Safari 浏览器浏览包含 Email 输入框的网页，Safari 浏览器会通过改变触摸屏键盘来配合该输入框，在触摸屏键盘中添加“@”和“.”键以方便用户输入，如图 3.3 所示，而当浏览普通内容时则不会出现这两个键。email 类型的 input 元素这一新增功能虽然用户不易察觉，但屏幕键盘的变化无疑会带来很好的用户体验。





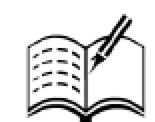

Note



图 3.3　iPhone 中的 Safari 浏览器触摸屏键盘随输入域改变而改变


视 频 讲 解

3.2.2　url——URL 地址框

url 类型的 input 元素提供用于输入 url 地址的文本框。当提交表单时，如果所输入的是 url 地址格式的字符串，则会提交服务器，如果不是，则不允许提交。

【示例】下面是 url 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入网址：<input type="url" name="user_url" /><br/>
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.4 所示。如果输入了错误的 url 地址格式，单击“提交”按钮时会出现如图 3.5 所示的“请输入网址”提示。

注意：www.baidu.com 并不是有效的 URL，因为 URL 必须以 http://或 https://开头。这里最好使用占位符提示访问者。另外，还可以在该字段下面的解释文本中指出合法的格式。

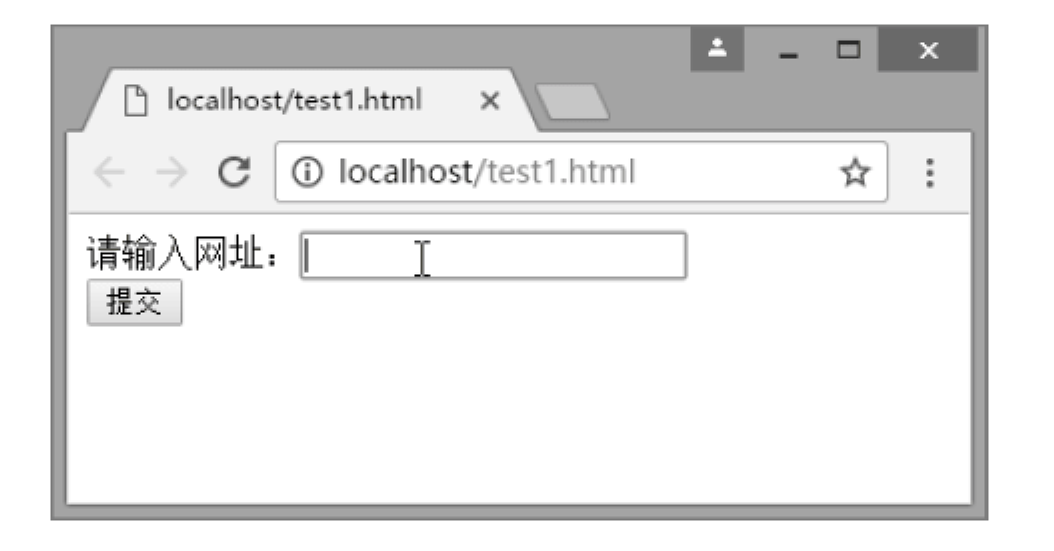


图 3.4　url 类型的 input 元素示例



图 3.5　检测到不是有效的 url 地址

与前面介绍的 email 类型输入框相同，对于不支持 type="url"的浏览器，将会以 type="text"来处理，所以并不妨碍旧版浏览器正常采用 type="url"输入框中的 URL 信息。

如果使用 iPhone 或 iPad 中的 Safari 浏览器浏览包含 url 输入域的网页，Safari 浏览器会通过改变触摸屏键盘来配合该输入框，在触摸屏键盘中添加“.”“/”“.com”键以方便用户输入，如图 3.6 所示，而当浏览普通内容时则不会出现这 3 个键。





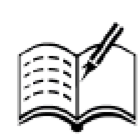




图 3.6　iPhone 中的 Safari 浏览器触摸屏键盘随输入域改变而改变



3.2.3　number——数字框

number 类型的 input 元素提供用于输入数值的文本框。用户还可以设定对所接受的数字的限制，包括允许的最大值和最小值、合法的数字间隔或默认值等。如果所输入的数字不在限定范围之内，则会提示错误信息。

【示例】下面是 number 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入数值：<input type="number" name="number1" min="1" max="20" step="4">
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.7 所示。如果输入了不在限定范围之内的数字，单击“提交”按钮时会出现如图 3.8 所示的提示。

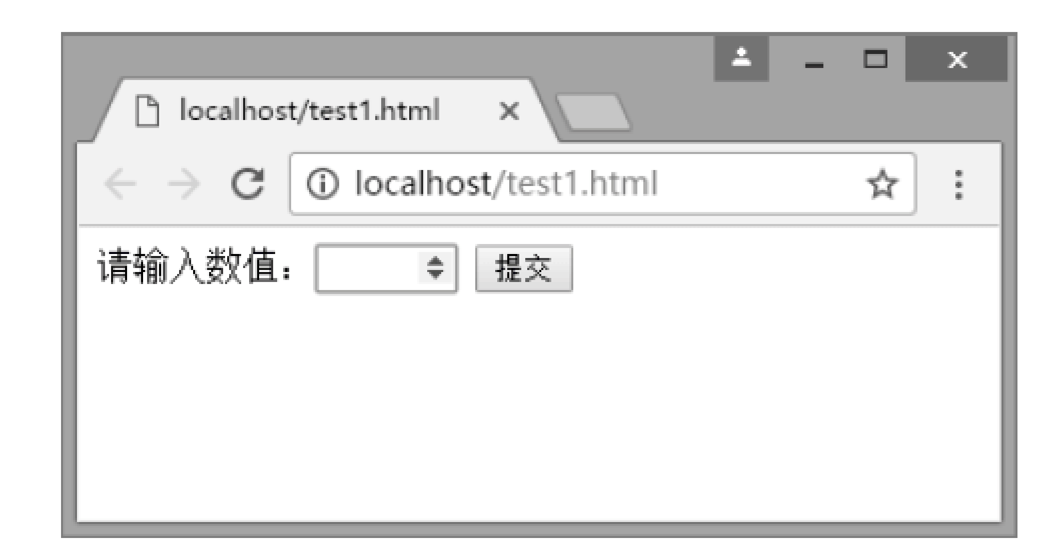


图 3.7　number 类型的 input 元素示例

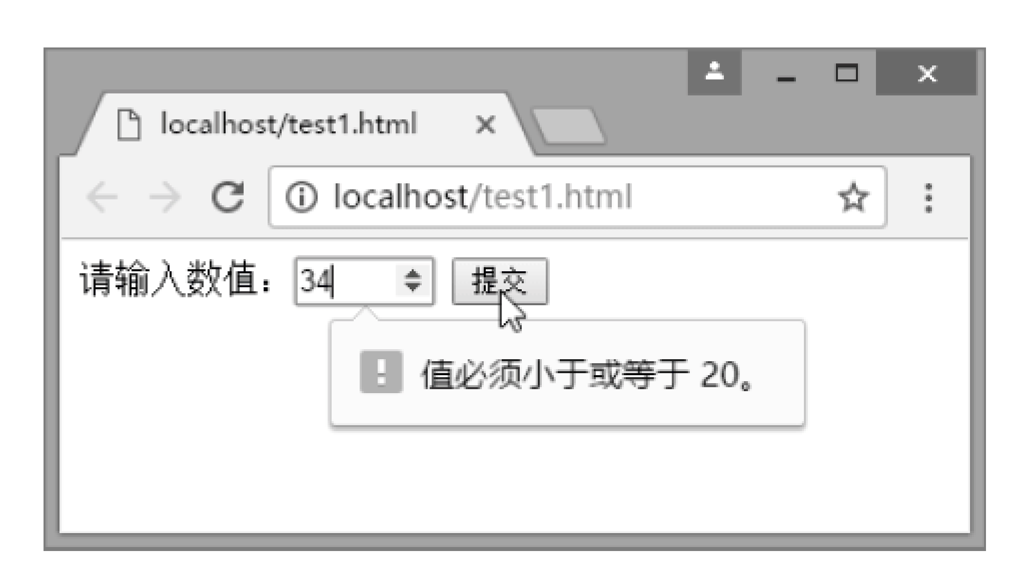


图 3.8　检测到输入了不在限定范围之内的数字

图 3.8 所示为输入了大于规定的最大值时所出现的提示。同样的，如果违反了其他限定，也会出现相关提示。例如，如果输入数值 15，则单击“提交”按钮时会出现如图 3.9 所示的提示。这是因为限定了合法的数字间隔为 4，在输入时只能输入 4 的倍数，如 4、8、16 等。又如，如果输入数值-12，则会提示“值必须大于或等于 1”，如图 3.10 所示。



Note

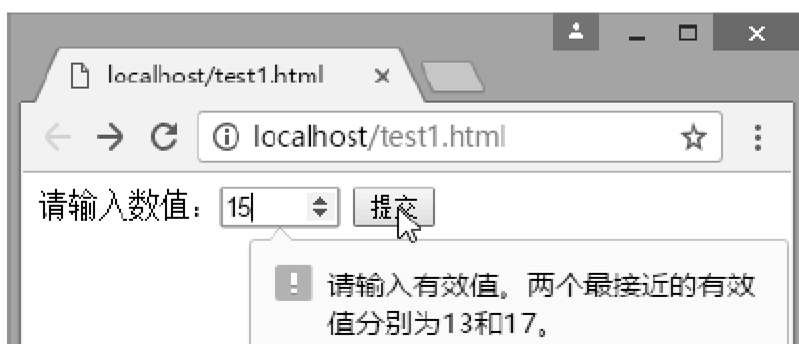
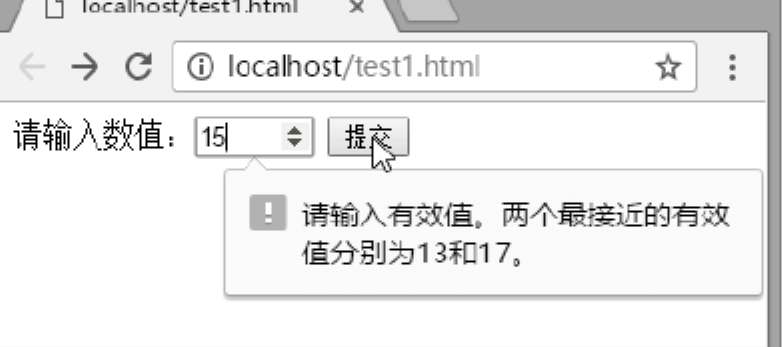



图 3.9　出现“值无效”的提示

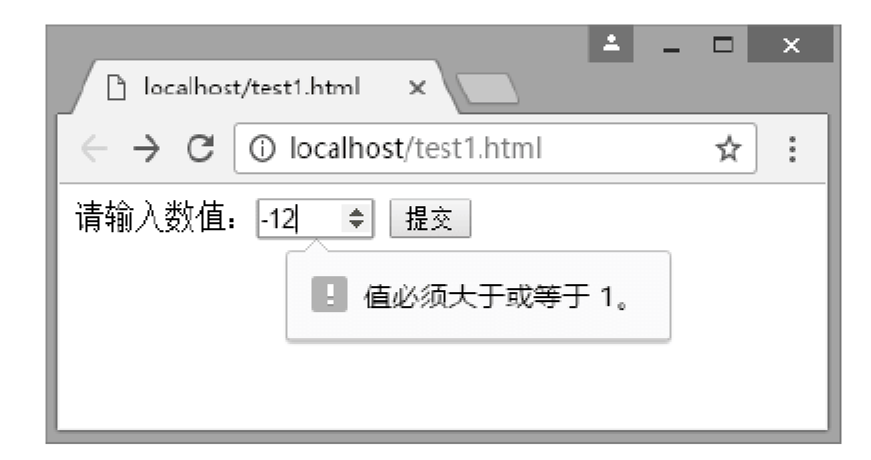


图 3.10　提示“值必须大于或等于 1”

number 类型使用下面的属性来规定对数字类型的限定，说明如表 3.1 所示。

表 3.1　number 类型的属性

属　　性	值	描　　述
max	number	规定允许的最大值
min	number	规定允许的最小值
step	number	规定合法的数字间隔（如果step="4"，则合法的数是-4、0、4、8 等）
value	number	规定默认值

对于不同的浏览器，number 类型的输入框的外观也可能会有所不同。而如果使用 iPhone 或 iPod 中的 Safari 浏览器浏览包含 number 输入框的网页，则 Safari 浏览器同样会通过改变触摸屏键盘来配合该输入框，触摸屏键盘会优化显示数字以方便用户输入，如图 3.11 所示。





图 3.11　iPhone 中的 Safari 浏览器触摸屏键盘显示出数字与符号



视频讲解

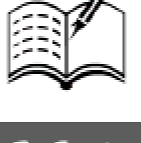
3.2.4　range——范围框

range 类型的 input 元素提供用于输入包含一定范围内数字值的文本框，在网页中显示为滑动条。用户可以设定对所接受的数字的限制，包括规定允许的最大值和最小值、合法的数字间隔或默认值等。如果所输入的数字不在限定范围之内，则会出现错误提示。

【示例】下面是 range 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入数值：  <input type="range" name="range1" min="1" max="30" />
<input type="submit" />
</form>
```





Note

以上代码在 Chrome 浏览器中的运行结果如图 3.12 所示。range 类型的 input 元素在不同浏览器中的外观也不同，例如在 Opera 浏览器中的外观如图 3.13 所示，会在滑块下方显示出额外的数字间隔短线。

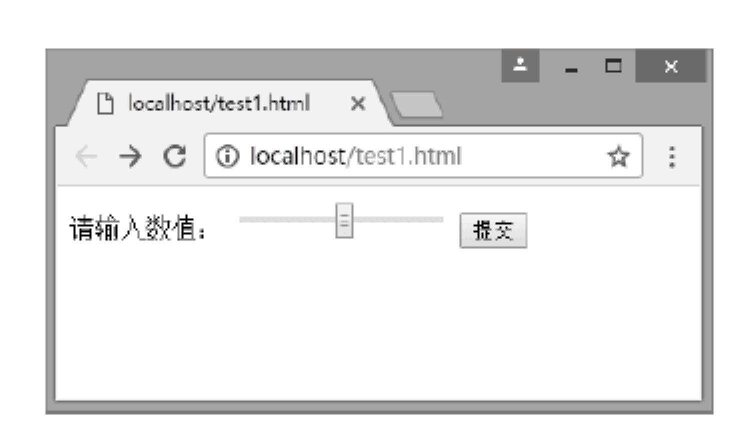


图 3.12　range 类型的 input 元素示例

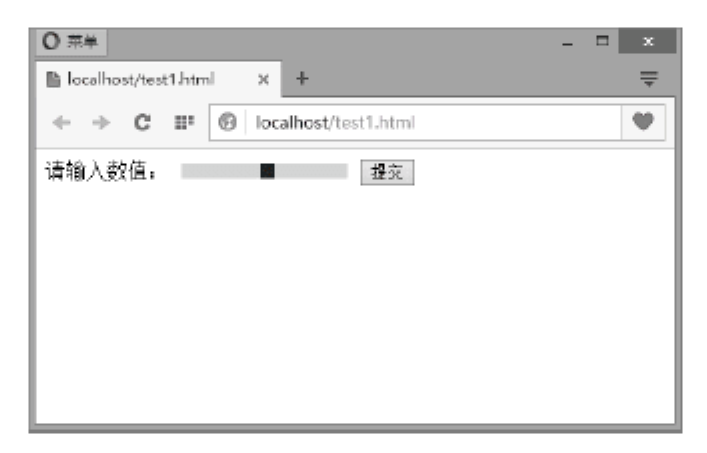


图 3.13　range 类型的 input 元素在 Opera 浏览器中的外观

range 类型使用下面的属性来规定对数字类型的限定，说明如表 3.2 所示。

表 3.2　range 类型的属性

属　性	值	描　述
max	number	规定允许的最大值
min	number	规定允许的最小值
step	number	规定合法的数字间隔（如果 step="4"，则合法的数是-4、0、4、8 等）
value	number	规定默认值

从表 3.2 中可以看出，range 类型的属性与 number 类型的属性相同，这两种类型的不同在于外观表现上，支持 range 类型的浏览器都会将其显示为滑块的形式，而不支持 range 类型的浏览器则会将其显示为普通的文本框，即以 type="text"来处理。

3.2.5　date pickers——日期选择器



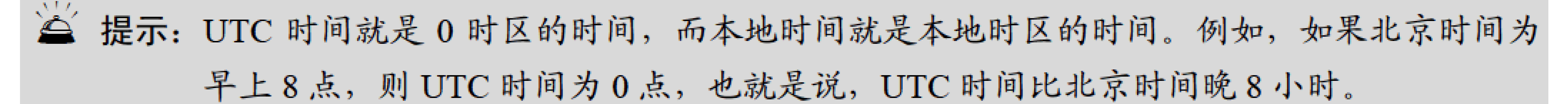
视频讲解

日期选择器（Date Pickers）是网页中经常要用到的一种控件，在 HTML5 之前的版本中，并没有提供任何形式的日期选择器控件，多采用一些 JavaScript 框架来实现日期选择器控件的功能，如 jQuery UI、YUI 等，在具体使用时会比较麻烦。

HTML5 提供了多个可用于选取日期和时间的输入类型，即 6 种日期选择器控件，分别用于选择以下日期格式：日期、月、星期、时间、日期+时间、日期+时间+时区，如表 3.3 所示。

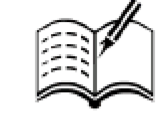
表 3.3　日期选择器类型

输 入 类 型	HTML 代码	功能与说明
date	<input type="date">	选取日、月、年
month	<input type="month">	选取月、年
week	<input type="week">	选取周和年
time	<input type="time">	选取时间（小时和分钟）
datetime	<input type="datetime">	选取时间、日、月、年（UTC 时间）
datetime-local	<input type="datetime-local">	选取时间、日、月、年（本地时间）



提示：UTC 时间就是 0 时区的时间，而本地时间就是本地时区的时间。例如，如果北京时间为早上 8 点，则 UTC 时间为 0 点，也就是说，UTC 时间比北京时间晚 8 小时。





Note

1. date 类型

date 类型的日期选择器用于选取日、月、年，即选择一个具体的日期，例如 2018 年 8 月 8 日，选择后会以 2018-08-08 的形式显示。

【示例 1】下面是 date 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入日期：  <input type="date" name=" date1" />
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.14 所示，在 Opera 浏览器中的运行结果如图 3.15 所示。Chrome 浏览器中显示为右侧带有微调按钮的数字输入框，可见该浏览器并不支持日期选择器控件。而 Opera 浏览器中单击右侧小箭头时会显示出日期控件，用户可以使用控件来选择具体日期。

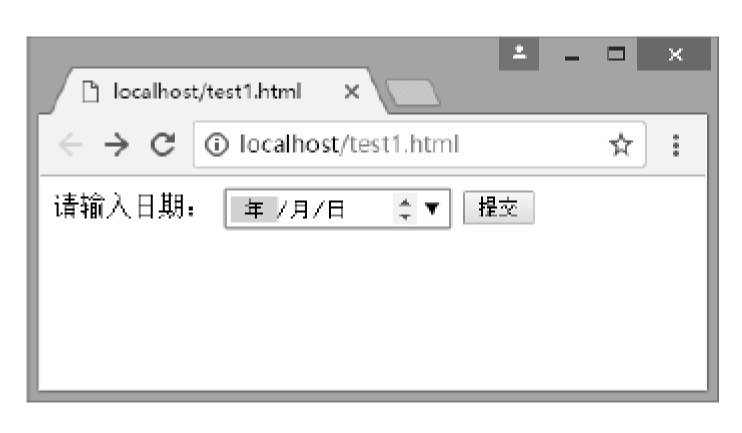


图 3.14　在 Chrome 浏览器中的运行结果

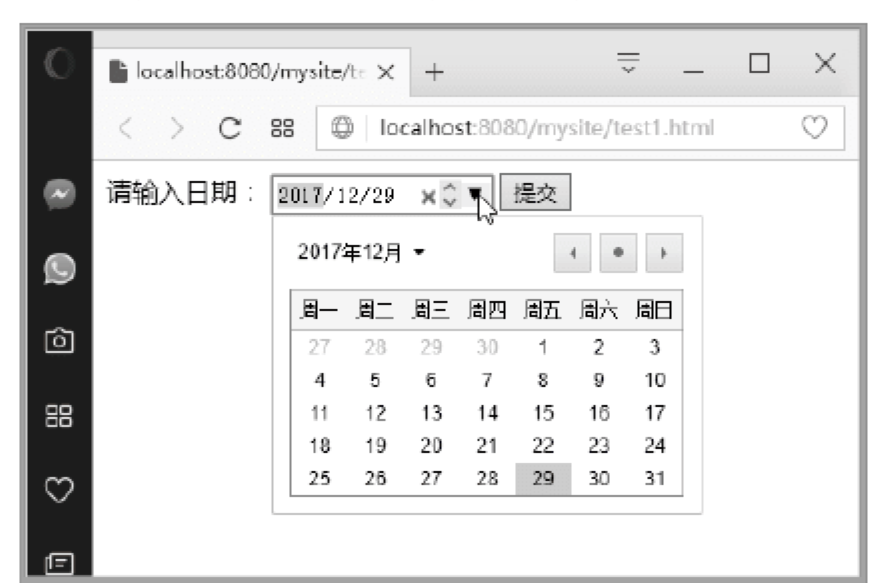


图 3.15　在 Opera 浏览器中的运行结果

2. month 类型

month 类型的日期选择器用于选取月、年，即选择一个具体的月份，例如 2018 年 8 月，选择后会以 2018-08 的形式显示。

【示例 2】下面是 month 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入月份：  <input type="month" name=" month1" />
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.16 所示，在 Opera 浏览器中的运行结果如图 3.17 所示。Chrome 浏览器中显示为右侧带有微调按钮的数字输入框，输入或微调时会只显示到月份，而不会显示日期。Opera 浏览器中单击右侧小箭头时会显示出日期控件，用户可以使用控件来选择具体月份，但不能选择具体日期。可以看到，整个月份中的日期都会以深灰色显示，单击该区域可以选择整个月份。

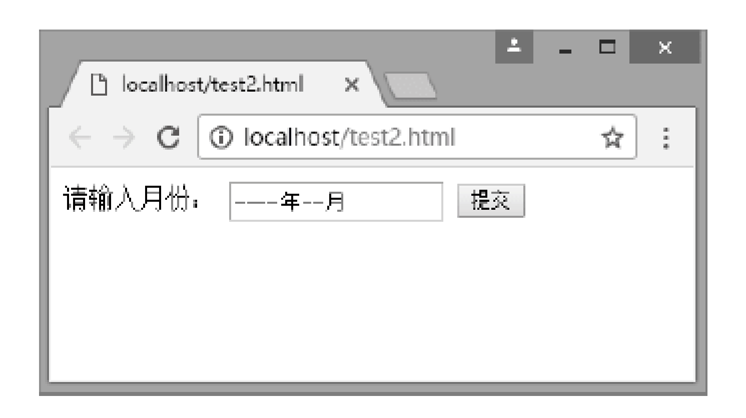


图 3.16　在 Chrome 浏览器中的运行结果



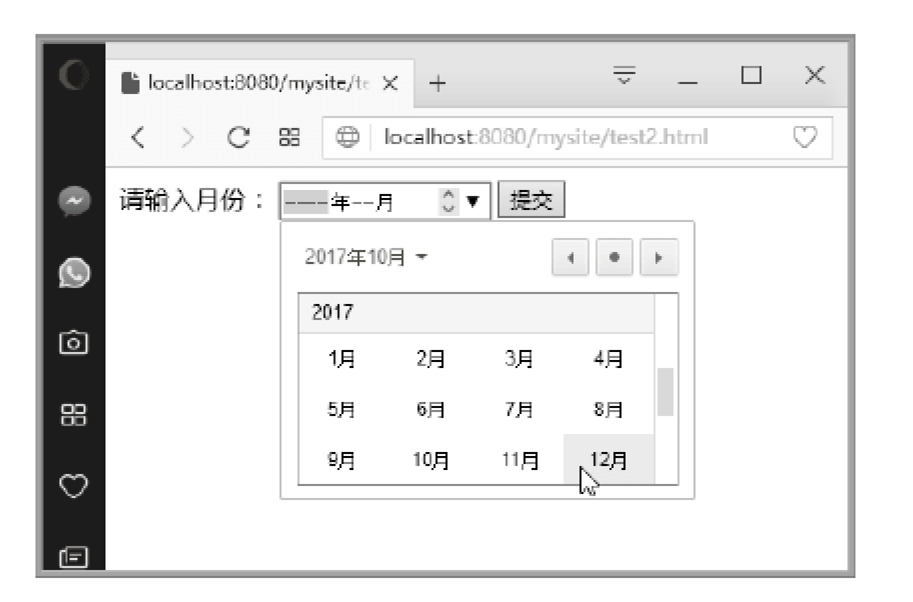


图 3.17　在 Opera 浏览器中的运行结果



3. week 类型

week 类型的日期选择器用于选取周和年，即选择一个具体的哪一周，例如 2017 年 10 月第 42 周，选择后会以“2017 年第 42 周”的形式显示。

【示例 3】下面是 week 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请选择年份和周数：  <input type="week" name="week1" />
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.18 所示，在 Opera 浏览器中的运行结果如图 3.19 所示。Chrome 浏览器中显示为右侧带有微调按钮的数字输入框，输入或微调时会显示年份和周数，而不会显示日期。Opera 浏览器中单击右侧小箭头时会显示出日期控件，用户可以使用控件来选择具体的年份和周数，但不能选择具体日期。可以看到，整个月份中的日期都会以深灰色按周数显示，单击该区域可以选择某一周。

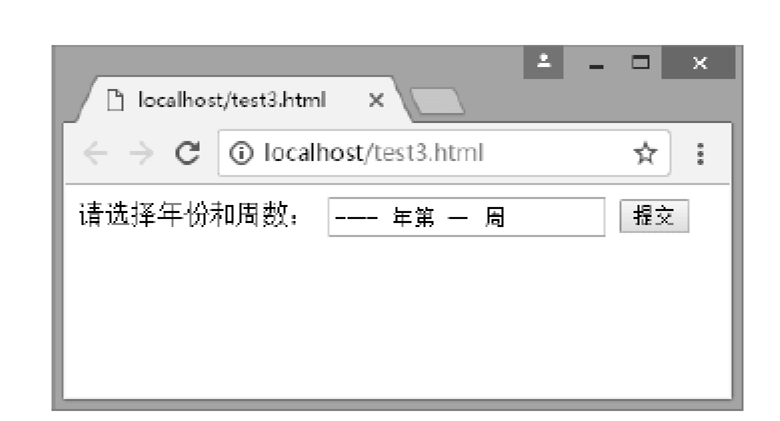


图 3.18　在 Chrome 浏览器中的运行结果

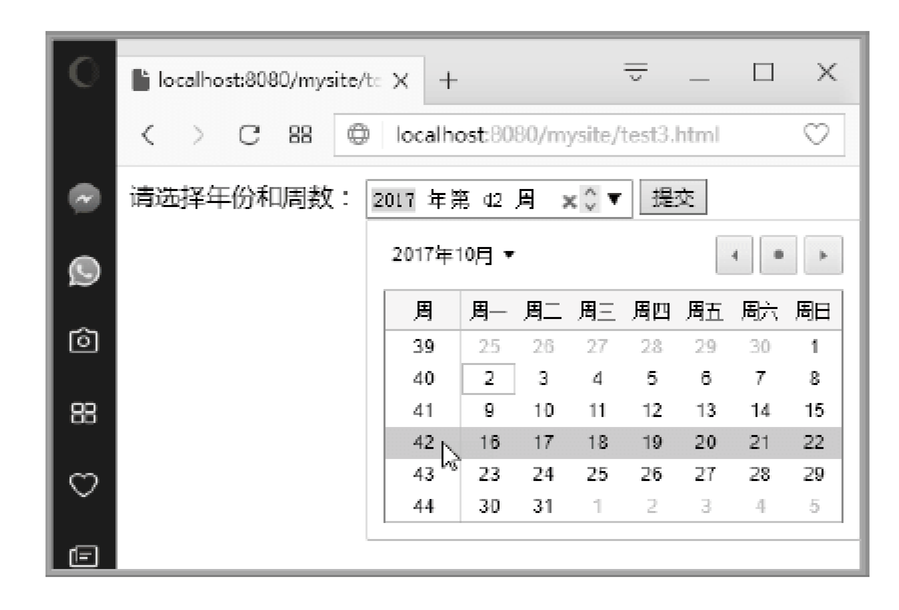


图 3.19　在 Opera 浏览器中的运行结果

4. time 类型

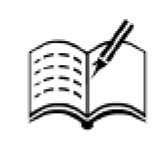
time 类型的日期选择器用于选取时间，具体到小时和分钟，例如，选择后会以 22:59 的形式显示。

【示例 4】下面是 time 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请选择或输入时间：  <input type="time" name="time1" />
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.20 所示，在 Opera 浏览器中的运行结果如图 3.21 所示。





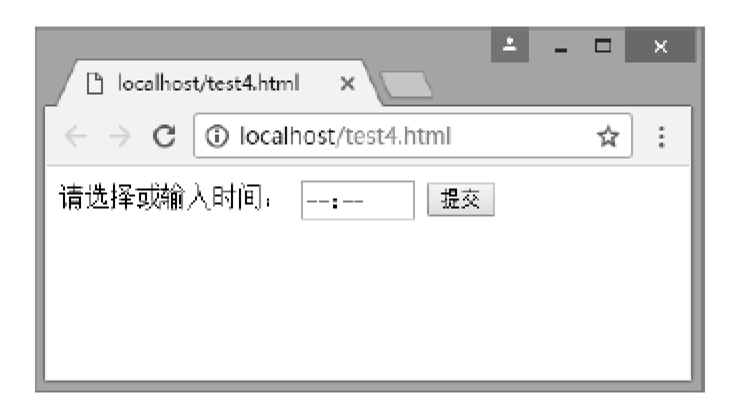




图 3.20　在 Chrome 浏览器中的运行结果

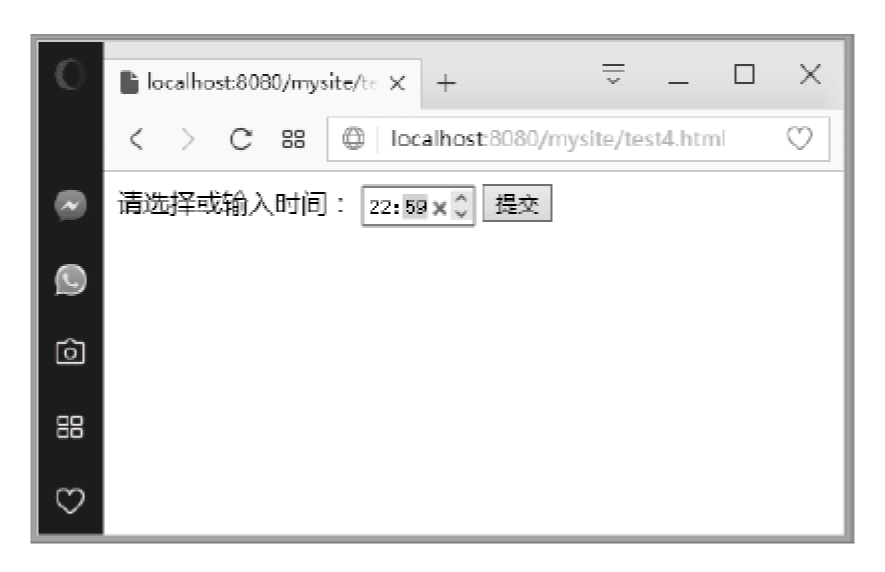


图 3.21　在 Opera 浏览器中的运行结果

除了可以使用微调按钮之外，还可以直接输入时间值。如果输入了错误的时间格式并单击了“提交”按钮，则在 Chrome 浏览器中会自动更正为最接近的合法值，而在 IE 10 浏览器中则以普通的文本框显示，如图 3.22 所示。

time 类型支持使用一些属性来限定时间的大小范围或合法的时间间隔，如表 3.4 所示。

表 3.4　time 类型的属性

属　　性	值	描　　述
max	time	规定允许的最大值
min	time	规定允许的最小值
step	number	规定合法的时间间隔
value	time	规定默认值

【示例 5】可以使用下列代码来限定时间。

```
<form action="demo_form.php" method="get">
请选择或输入时间：<input type="time" name="time1" step="5" value="09:00">
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.23 所示，可以看到，在输入框中出现设置的默认值“09:00”，并且当单击微调按钮时，会以 5 秒钟为单位递增或递减。当然，用户还可以使用 min 和 max 属性指定时间的范围。



图 3.22　IE10 不支持该类型输入框

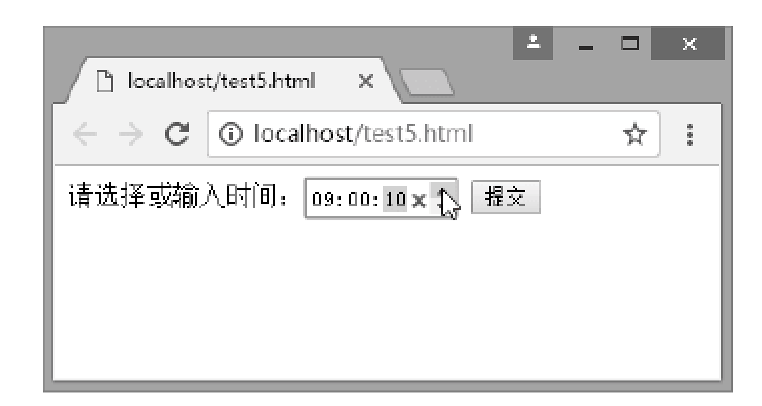


图 3.23　使用属性值限定时间类型

在 date 类型、month 类型、week 类型中也支持使用上述属性值。

5. datetime 类型

datetime 类型的日期选择器用于选取时间、日、月、年，其中时间为 UTC 时间。

【示例 6】下面是 datetime 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
```



```
请选择或输入时间：<input type="datetime" name="datetime1" />
<input type="submit" />
</form>
```





以上代码在 Safari 浏览器中的运行结果如图 3.24 所示，在 iPhone 中的运行结果如图 3.25 所示。

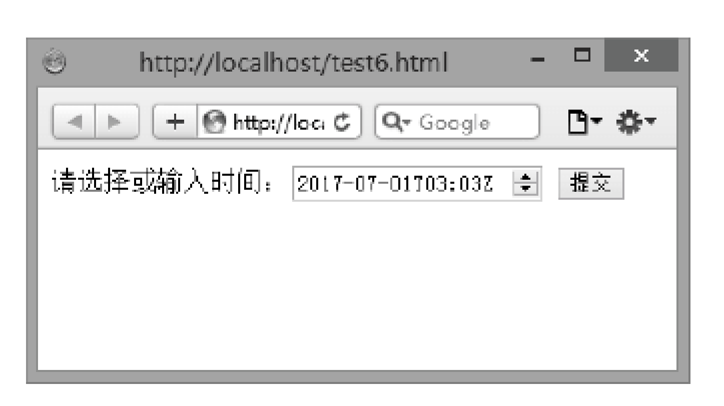


图 3.24　在 Safari 浏览器中的运行结果



图 3.25　在 iPhone 中的运行结果

注意：IE、Firefox 和 Chrome 不再支持<input type="datetime">元素，Chrome 和 Safari 部分版本支持。Opera 12 以及更早的版本完全支持。

6. datetime-local 类型

datetime-local 类型的日期选择器用于选取时间、日、月、年，其中时间为本地时间。

【示例 7】下面是 datetime-local 类型的一个应用示例。

```
<form action="demo form.php" method="get">
请选择或输入时间：<input type="datetime-local" name="datetime-local1" />
<input type="submit" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.26 所示，在 Opera 浏览器中的运行结果如图 3.27 所示。

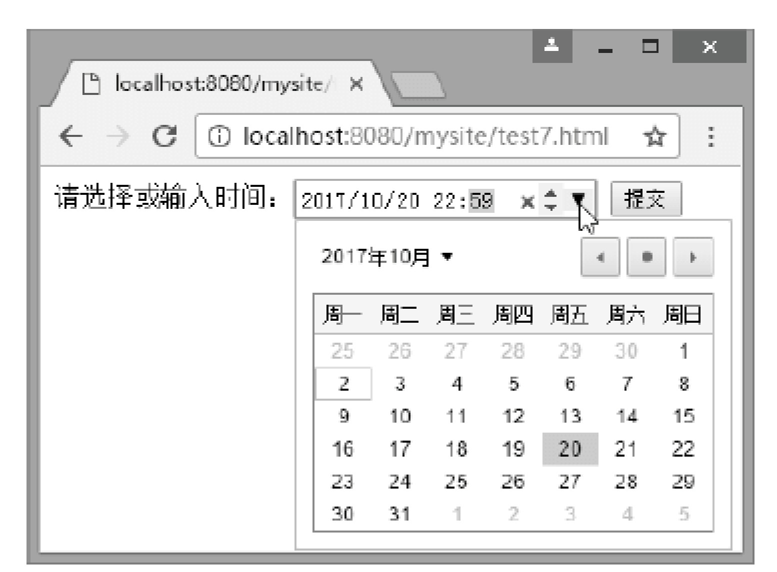


图 3.26　在 Chrome 浏览器中的运行结果

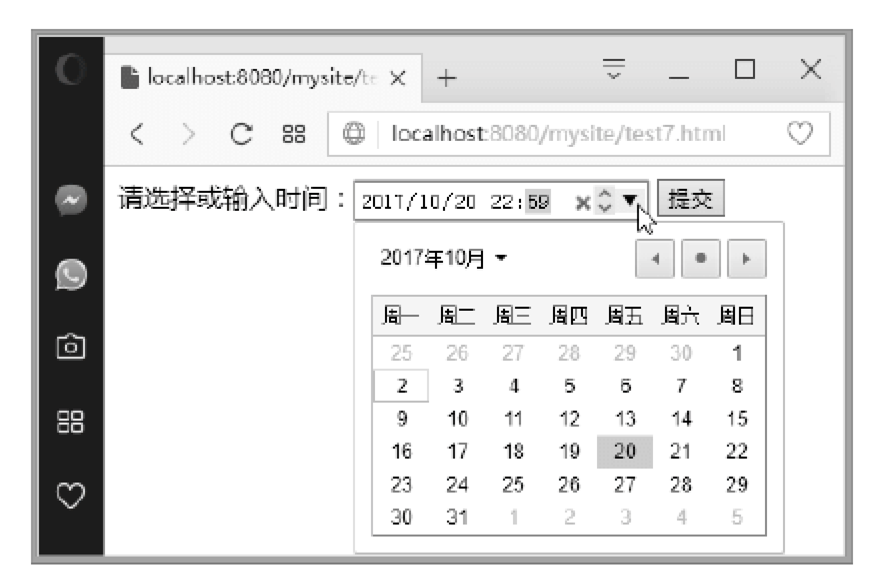
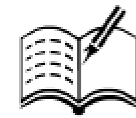


图 3.27　在 Opera 浏览器中的运行结果



视频讲解



Note

3.2.6 search——搜索框

search 类型的 input 元素提供用于输入搜索关键词的文本框。在外观上看起来，search 类型的 input 元素与普通的 text 类型的区别：当输入内容时，右侧会出现一个“×”图标，单击即可清除搜索框。

【示例】下面是 search 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入搜索关键词：<input type="search" name="search1" />
<input type="submit" value="Go"/>
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.28 所示。如果在搜索框中输入要搜索的关键词，在搜索框右侧就会出现一个“×”按钮。单击该按钮可以清除已经输入的内容。在 Windows 系统中，新版的 IE、Chrome、Opera 浏览器支持“×”按钮这一功能，Firefox 浏览器则不支持，如图 3.29 所示。

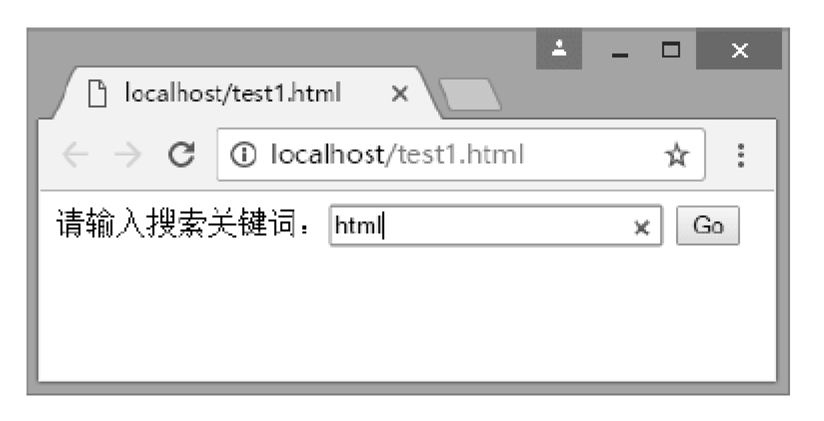


图 3.28　search 类型的应用

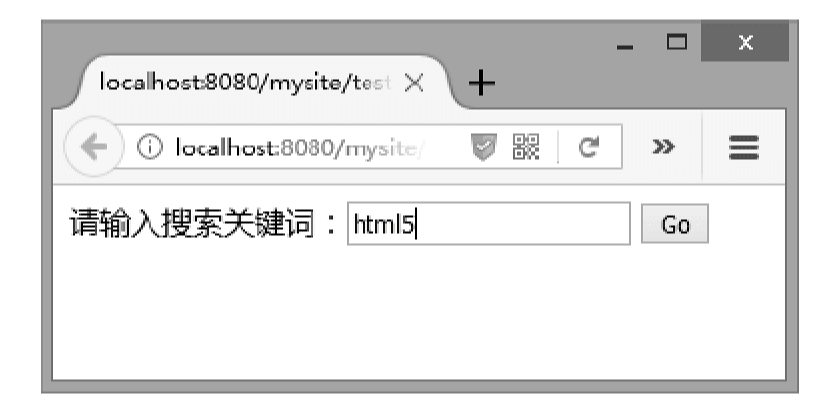


图 3.29　Firefox 没有“×”按钮

在 Mac OS X 或 iOS 系统中，Safari 浏览器会将搜索框渲染成圆角，如图 3.30 所示，而不是 Windows 系统中用户常见到的方角。

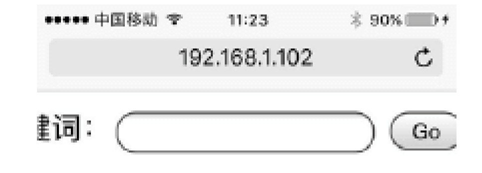




图 3.30　iOS 系统中的圆角搜索框

提示：在默认情况下，为 Chrome、Safari 和 Mobile Safari 等浏览器中的搜索框设置样式是受到限制的。如果要消除这一约束，重新获得 CSS 的控制权，可以使用专有的“-webkit-appearance: none;”声明，例如:

```
input[type="search"] {
    -webkit-appearance: none;
}
```

注意：appearance 属性并不是官方的 CSS，因此不同浏览器的行为有可能不一样。





视 频 讲 解

Note

3.2.7　tel——电话号码框

tel 类型的 input 元素提供专门用于输入电话号码的文本框。它并不限定只输入数字，因为很多的电话号码还包括其他字符，如“+”“-”“(”“)”等，例如 86-0536-8888888。

【示例】下面是 tel 类型的一个应用示例。

```
<form action="demo_form.php" method="get">
请输入电话号码：<input type="tel" name="tel1" />
<input type="submit" value="提交"/>
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.31 所示。从某种程度上来说，所有的浏览器都支持 tel 类型的 input 元素，因为它们都会将其作为一个普通的文本框来显示。HTML5 规则并不需要浏览器执行任何特定的电话号码语法或以任何特别的方式来显示电话号码。

iPhone 或 iPad 中的浏览器遇到 tel 类型的 input 元素时，会自动变换触摸屏幕键盘以方便用户输入，如图 3.32 所示。

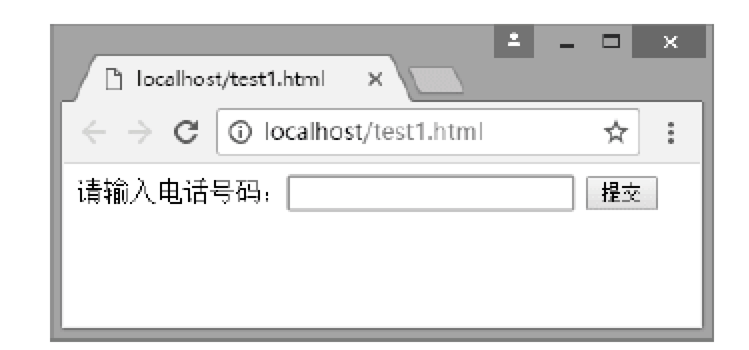


图 3.31　tel 类型的应用



图 3.32　iPhone 中的屏幕键盘变化

3.2.8　color——拾色器

视 频 讲 解

color 类型的 input 元素提供专门用于选择颜色的文本框。当 color 类型文本框获取焦点后，会自动调用系统的颜色窗口，包括苹果系统也能弹出相应的系统色盘。

【示例】下面是 color 类型的一个应用示例。

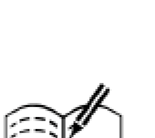
```
<form action="demo_form.php" method="get">
请选择一种颜色：<input type="color" name="color1" />
<input type="submit" value="提交"/>
</form>
```

以上代码在 Opera 浏览器中的运行结果如图 3.33 所示，单击颜色文本框，会打开 Windows 系统的“颜色”对话框，如图 3.34 所示，选择一种颜色之后，单击“确定”按钮返回网页，这时可以看到颜色文本框显示对应颜色效果，如图 3.35 所示。



提示：IE 和 Safari 浏览器暂不支持，Mac OS 和 iOS 系统也不支持。





Note



图 3.33　color 类型的应用

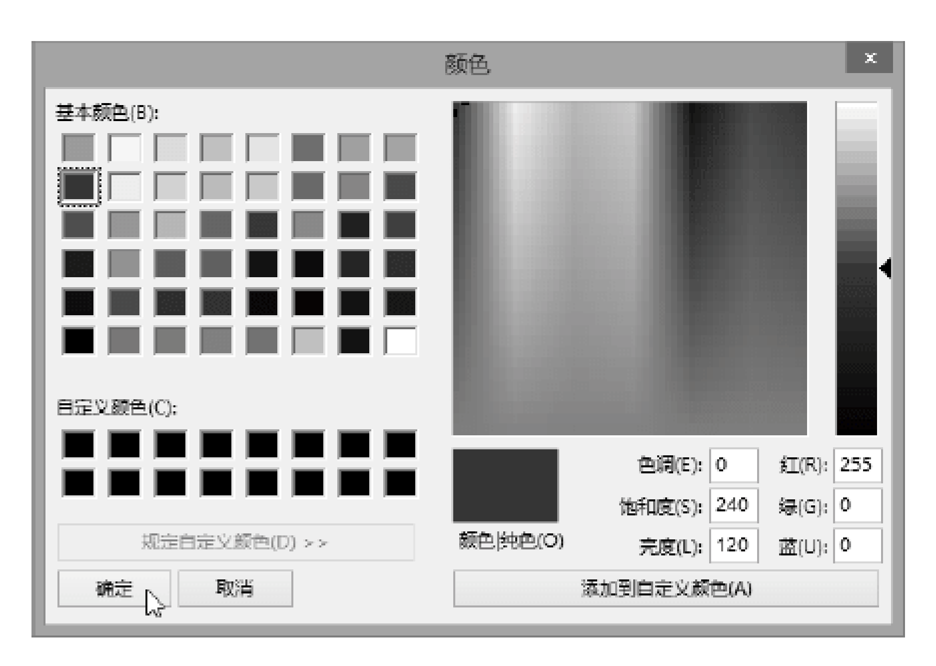


图 3.34　Windows 系统中的“颜色”对话框

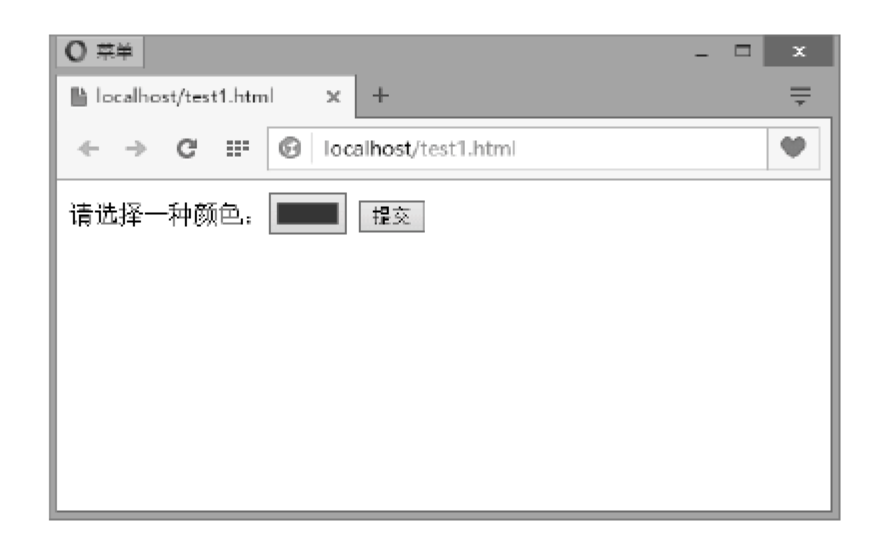


图 3.35　设置颜色后效果

3.3　新的 input 属性

HTML5 为 input 元素新增了多个属性，用于限制输入行为或格式。



视频讲解

3.3.1　autocomplete——自动完成

autocomplete 属性可以帮助用户在输入框中实现自动完成输入。取值包括 on 和 off，用法如下所示。

```
<input type="email" name="email" autocomplete="off" />
```

提示：autocomplete 属性适用 input 类型包括：text、search、url、telephone、email、password、datepickers、range 和 color。

autocomplete 属性也适用于 form 元素，默认状态下表单的 autocomplete 属性处于打开状态，其包含的输入域会自动继承 autocomplete 状态，也可以为某个输入域单独设置 autocomplete 状态。

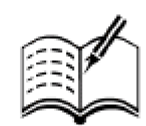
注意：在某些浏览器中需要先启用浏览器本身的自动完成功能，才能使 autocomplete 属性起作用。

【示例】设置 autocomplete 为 on 时，可以使用 HTML5 新增的 datalist 元素和 list 属性提供一个数据列表供用户进行选择。下面示例演示如何应用 autocomplete 属性、datalist 元素和 list 属性实现自动完成。

```
<h2>输入你最喜欢的城市名称</h2>
<form autocompelete="on">
    <input type="text" id="city" list="cityList">
```



```
        <datalist id="cityList" style="display:none;">
            <option value="BeiJing">BeiJing</option>
            <option value="QingDao">QingDao</option>
            <option value="QingZhou">QingZhou</option>
            <option value="QingHai">QingHai</option>
        </datalist>
    </form>
```



Note

在浏览器中预览，当用户将焦点定位到文本框中时，会自动出现一个城市列表供用户选择，如图 3.36 所示。而当用户单击页面的其他位置时，这个列表就会消失。

当用户输入时，该列表会随用户的输入自动更新，例如，当输入字母 q 时，会自动更新列表，只列出以 q 开头的城市名称，如图 3.37 所示。随着用户不断地输入新的字母，下面的列表还会随之变化。

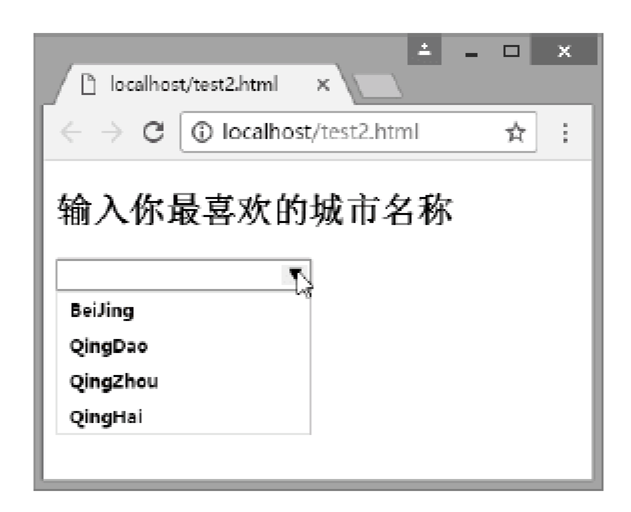


图 3.36　自动完成数据列表

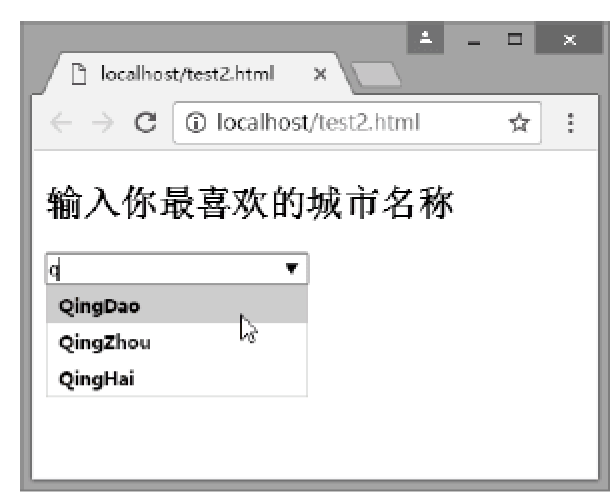


图 3.37　数据列表随用户输入而更新

提示：多数浏览器都带有辅助用户完成输入的自动完成功能，只要开启了该功能，浏览器会自动记录用户所输入的信息，当再次输入相同的内容时，浏览器就会自动完成内容的输入。从安全性和隐私的角度考虑，这个功能存在较大的隐患，如果不希望浏览器自动记录这些信息，则可以为 form 或 form 中的 input 元素设置 autocomplete 属性，关闭该功能。

3.3.2　autofocus——自动获取焦点



视频讲解

autofocus 属性可以实现在页面加载时，让表单控件自动获得焦点。用法如下所示。

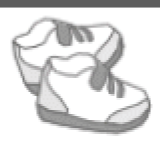
```
<input type="text" name="fname" autofocus="autofocus" />
```

autocomplete 属性适用所有<input>标签的类型，如文本框、复选框、单选按钮、普通按钮等。

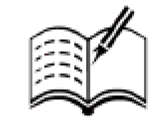
注意：在同一页面中只能指定一个 autofocus 对象，当页面中的表单控件比较多时，建议为最需要聚焦的那个控件设置 autofocus 属性值，如页面中搜索文本框，或者许可协议的“同意”按钮等。

【示例 1】下面示例演示如何应用 autofocus 属性。

```
<form>
    <p>请仔细阅读许可协议：</p>
    <p>
        <label for="textarea1"></label>
        <textarea name="textarea1" id="textarea1" cols="45" rows="5">许可协议具体内容......</textarea>
    </p>
    <p>
```



```
            <input type="submit" value="同意" autofocus>
            <input type="submit" value="拒绝">
        </p>
    </form>
```



Note

以上代码在 Chrome 浏览器中的运行结果如图 3.38 所示。页面载入后，“同意”按钮自动获得焦点，因为通常希望用户直接单击该按钮。如果将“拒绝”按钮的 autofocus 属性值设置为 on，则页面载入后焦点就会在“拒绝”按钮上，如图 3.39 所示，但从页面功用的角度来说这样并不合适。

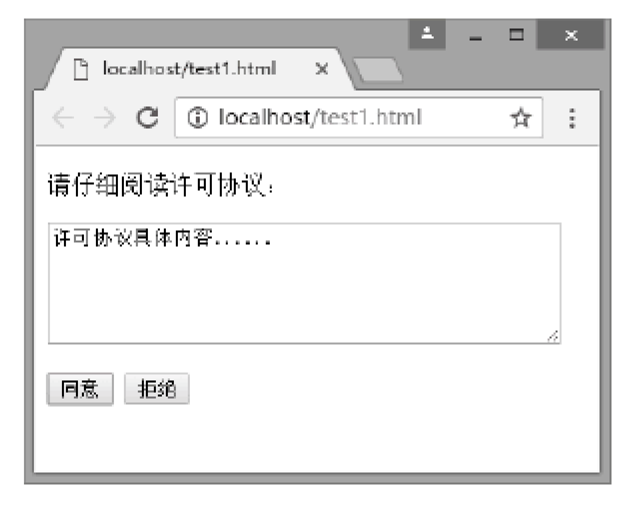


图 3.38　“同意”按钮自动获得焦点

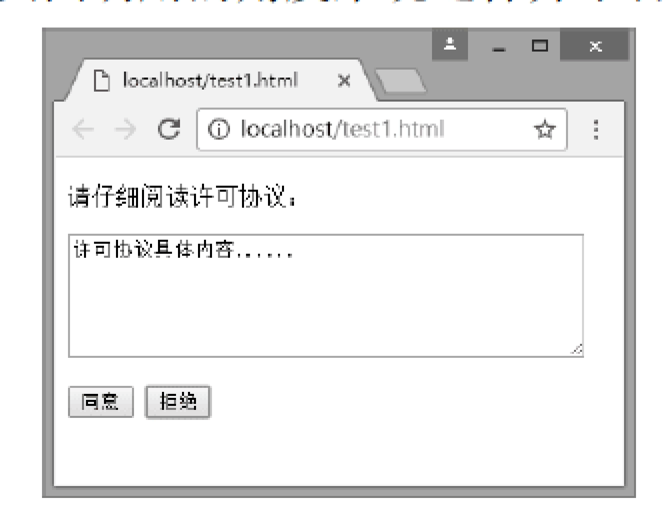


图 3.39　“拒绝”按钮自动获得焦点

【示例 2】如果浏览器不支持 autofocus 属性，可以使用 JavaScript 实现相同的功能。在下面脚本中，先检测浏览器是否支持 autofocus 属性，如果不支持则获取指定的表单域，为其调用 focus()方法，强迫其获取焦点。

```
<script>
if (!("autofocus" in document.createElement("input"))) {
    document.getElementById("ok").focus();
}
</script>
```



视频讲解

3.3.3　form——归属表单

form 属性可以设置表单控件归属的表单。适用于所有<input>标签的类型。

> 提示：在 HTML4 中，用户必须把相关的控件放在表单内部，即<form>和</form>之间。在提交表单时，在<form>和</form>之外的控件将被忽略。

【示例】form 属性必须引用所属表单的 id，如果一个 form 属性要引用两个或两个以上的表单，则需要使用空格将表单的 id 值分隔开。下面是一个 form 属性应用。

```
<form action="" method="get" id="form1">
请输入姓名：<input type="text" name="name1" autofocus/>
<input type="submit"   value="提交"/>
</form>
请输入住址：<input type="text" name="address1" form="form1" />
```

以上代码在 Chrome 浏览器中的运行结果如图 3.40 所示。如果填写姓名和住址并单击“提交”按钮，则 name1 和 address1 分别会被赋值为所填写的值。例如，如果在姓名处填写“zhangsan”，住址处填写“北京”，则单击“提交”按钮后，服务器端会接收到“name1=zhangsan”和“address1=北京”。用户也可以在提交后观察浏览器的地址栏，可以看到有“name1=zhangsan&address1=北京”字样，如图 3.41 所示。



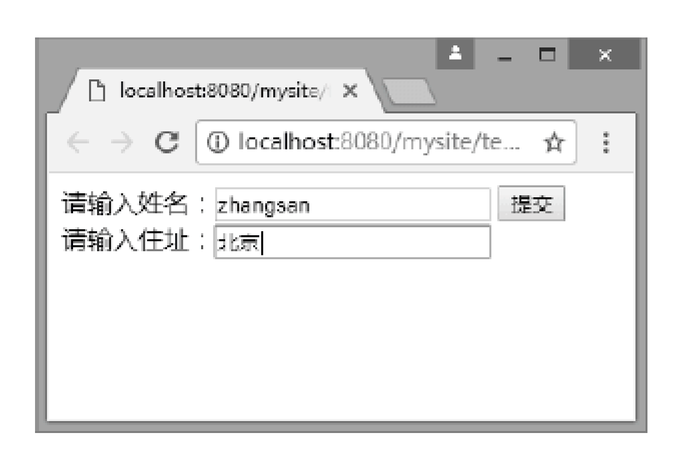


图 3.40　form 属性的应用

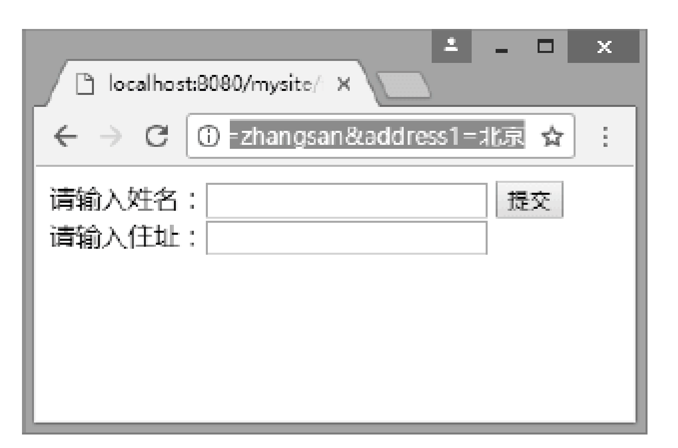


图 3.41　地址中要提交的数据

Note



3.3.4　表单重写

HTML5 新增 5 个表单重写属性，用于重写<form>标签属性设置，简单说明如下：

☑　formaction：重写<form>标签的 action 属性。

☑　formenctype：重写<form>标签的 enctype 属性。

☑　formmethod：重写<form>标签的 method 属性。

☑　formnovalidate：重写<form>标签的 novalidate 属性。

☑　formtarget：重写<form>标签的 target 属性。

注意：表单重写属性仅适用于 submit 和 image 类型的 input 元素。

【示例】下面示例设计通过 formaction 属性，实现将表单提交到不同的服务器页面。

```
<form action="1.asp" id="testform">
请输入电子邮件地址：<input type="email" name="userid" /><br />
    <input type="submit" value="提交到页面 1" formaction="1.asp" />
    <input type="submit" value="提交到页面 2" formaction="2.asp" />
    <input type="submit" value="提交到页面 3" formaction="3.asp" />
</form>
```



3.3.5　height 和 width——高和宽

height 和 width 属性仅用于设置<input type="image">标签的图像高度和宽度。

【示例】下面示例演示了 height 与 width 属性的应用。

```
<form action="testform.asp" method="get">
请输入用户名: <input type="text" name="user_name" /><br />
<input type="image" src="images/submit.png" width="72" height="26" />
</form>
```

源图像的大小为 288×104 像素，使用以上代码将其大小限制为 72×26 像素，在 Chrome 浏览器中的运行结果如图 3.42 所示。

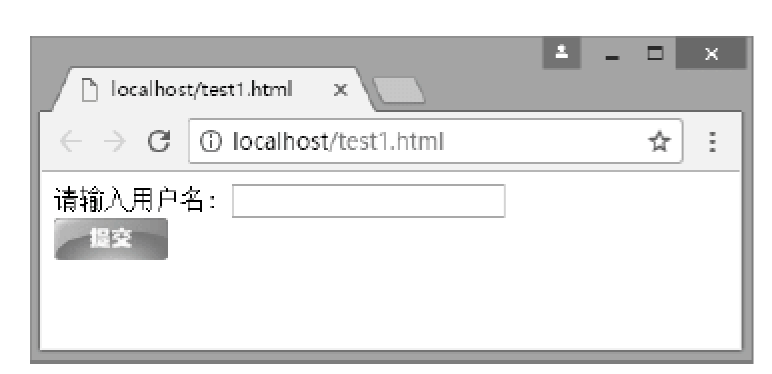
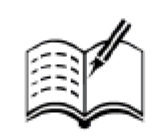


图 3.42　form 属性的应用



视 频 讲 解



Note



视 频 讲 解

3.3.6 list——列表选项

list 属性用于设置输入域的 datalist。datalist 是输入域的选项列表。该属性适用于以下类型的<input>标签：text、search、url、telephone、email、date pickers、number、range 和 color。

演示示例可参考 3.4.1 节 datalist 元素介绍。

注意：目前最新的主流浏览器都已支持 list 属性，不过呈现形式略有不同。

3.3.7 min、max 和 step——最小值、最大值和步长

min、max 和 step 属性用于为包含数字或日期的 input 输入类型设置限值，适用于 date pickers、number 和 range 类型的<input>标签。具体说明如下：

- ☑ min 属性：设置输入框所允许的最小值。
- ☑ max 属性：设置输入框所允许的最大值。
- ☑ step 属性：为输入框设置合法的数字间隔（步长）。例如，step="4"，则合法值包括-4、0、4 等。

【示例】下面示例设计一个数字输入框，并规定该输入框接受介于 0 到 12 的值，且数字间隔为 4。

```
<form action="testform.asp" method="get">
    请输入数值：<input type="number" name="number1" min="0" max="12" step="4" />
    <input type="submit" value="提交" />
</form>
```

在 Chrome 浏览器中运行，如果单击数字输入框右侧的微调按钮，则可以看到数字以 4 为步进值递增，如图 3.43 所示；如果输入不合法的数值，如 5，单击“提交”按钮时会显示错误提示，如图 3.44 所示。

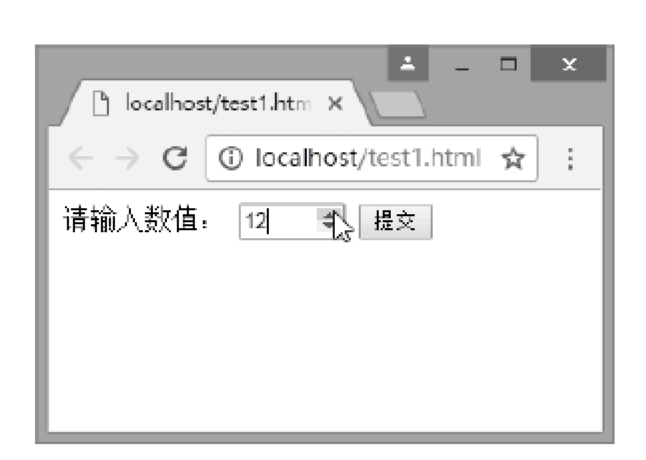


图 3.43　list 属性应用



图 3.44　显示错误提示



视 频 讲 解

3.3.8 multiple——多选

multiple 属性可以设置输入域一次选择多个值。适用于 email 和 file 类型的<input>标签。

【示例】下面在页面中插入一个文件域，使用 multiple 属性允许用户一次提交多个文件。

```
<form action="testform.asp" method="get">
    请选择要上传的多个文件：<input type="file" name="img" multiple />
    <input type="submit" value="提交" />
</form>
```



在 Chrome 浏览器中的运行结果如图 3.45 所示。如果单击“选择文件”按钮，则会允许在打开的对话框中选择多个文件。选择文件并单击“打开”按钮后会关闭对话框，同时在页面中会显示选中文件的个数，如图 3.46 所示。

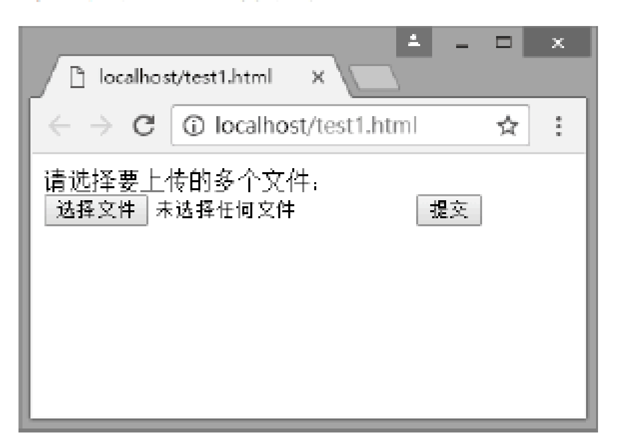
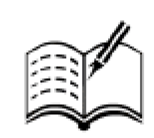


图 3.45　multiple 属性的应用



图 3.46　显示被选中文件的个数



3.3.9　pattern——匹配模式



pattern 属性规定用于验证 input 域的模式（pattern）。模式就是 JavaScript 正则表达式，通过自定义的正则表达式匹配用户输入的内容，以便进行验证。该属性适用于 text、search、url、telephone、email 和 password 类型的<input>标签。

> 提示：读者可以在 http://html5pattern.com 上面找到一些常用的正则表达式，并将它们复制、粘贴到自己的 pattern 属性中进行应用。

【示例】下面示例使用 pattern 属性设置文本框必须输入 6 位数的邮政编码。

```
<form action="/testform.asp" method="get">
    请输入邮政编码: <input type="text" name="zip_code" pattern="[0-9]{6}"
                                    title="请输入 6 位数的邮政编码" />
    <input type="submit" value="提交" />
</form>
```

在 Chrome 浏览器中的运行结果如图 3.47 所示。如果输入的数字不是 6 位，则会出现错误提示，如图 3.48 所示。如果输入的并非规定的数字，而是字母，也会出现这样的错误提示，因为 pattern="[0-9]{6}"中规定了必须输入 0～9 这样的阿拉伯数字，并且必须为 6 位数。

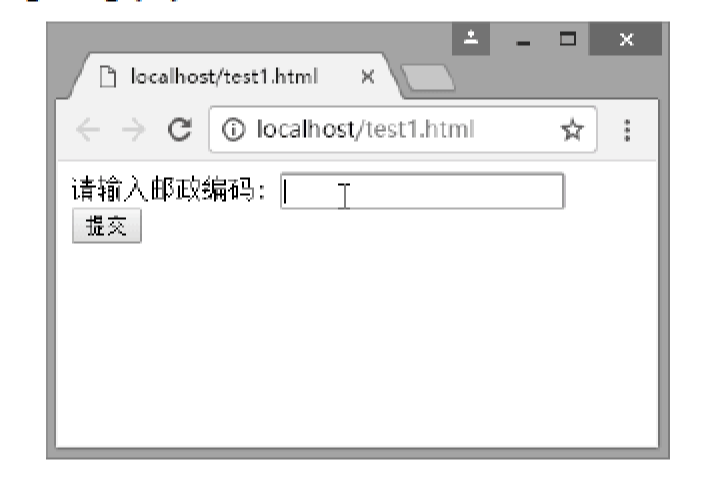


图 3.47　pattern 属性的应用

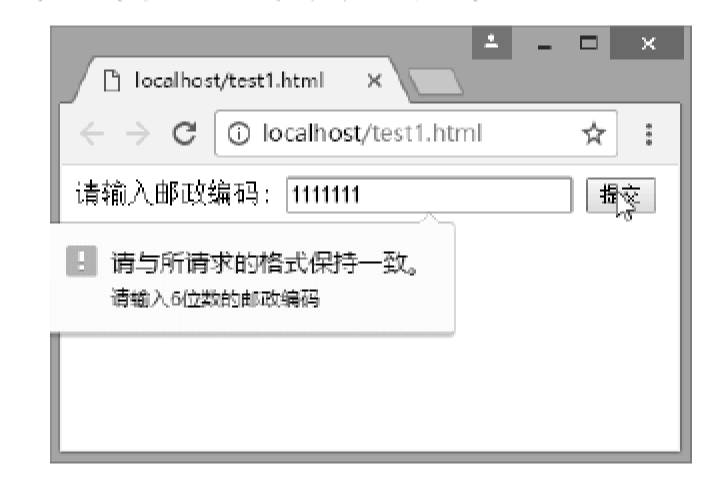


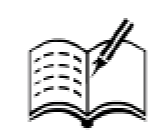
图 3.48　出现错误提示

3.3.10　placeholder——替换文本



placeholder 属性用于为 input 类型的输入框提供一种文本提示，这些提示可以描述输入框期待用户输入的内容，在输入框为空时显示，而当输入框获取焦点时自动消失。placeholder 属性适用于 text、search、url、telephone、email 和 password 类型的<input>标签。





Note

【示例】下面是 placeholder 属性的一个应用示例。请注意比较本例与上例提示方法的不同。

```
<form action="/testform.asp" method="get">
    请输入邮政编码:
    <input type="text" name="zip_code" pattern="[0-9]{6}"
placeholder="请输入 6 位数的邮政编码" />
    <input type="submit" value="提交" />
</form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.49 所示。当输入框获得焦点并输入字符时，提示文字消失，如图 3.50 所示。

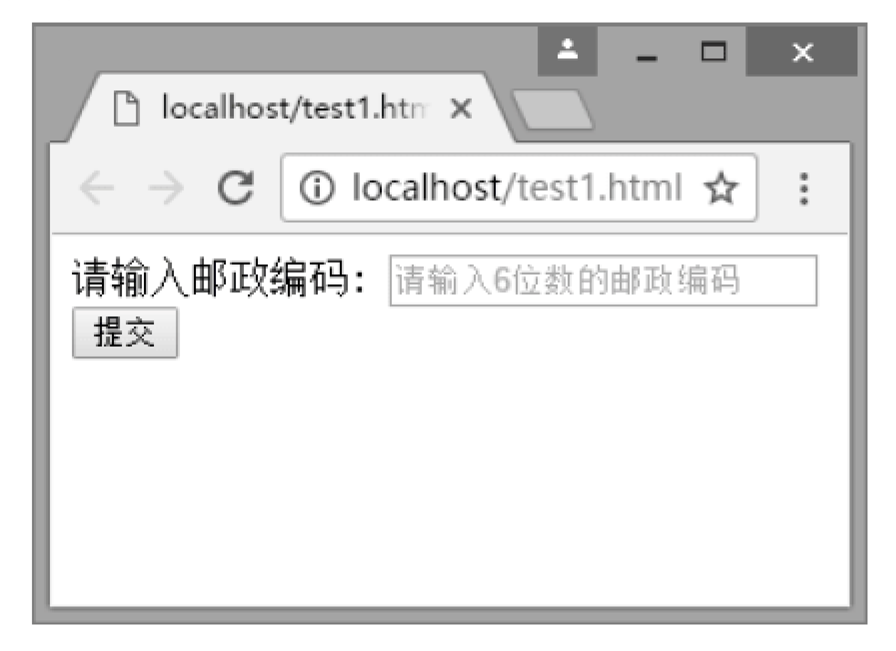


图 3.49　placeholder 属性的应用

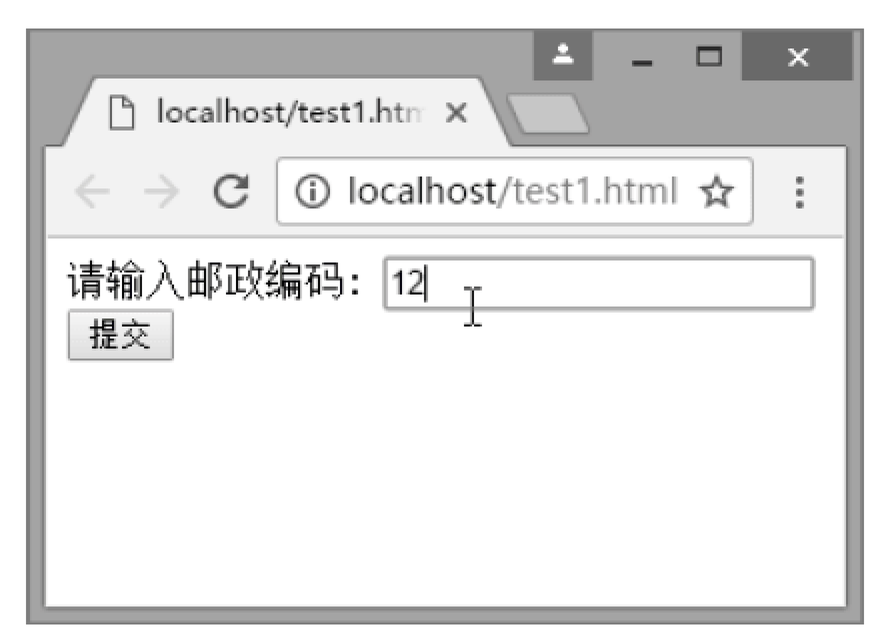


图 3.50　提示消失



视 频 讲 解

3.3.11　required——必填

required 属性用于定义输入框填写的内容不能为空，否则不允许提交表单。该属性适用于 text、search、url、telephone、email、password、date pickers、number、checkbox、radio 和 file 类型的<input>标签。

【示例】下面示例使用 required 属性规定文本框必须输入内容。

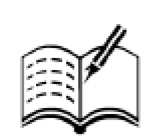
```
<form action="/testform.asp" method="get">
    请输入姓名: <input type="text" name="usr_name" required="required" />
    <input type="submit" value="提交" />
</form>
```

在 Chrome 浏览器中的运行结果如图 3.51 所示。当输入框内容为空并单击“提交”按钮时，会出现“请填写此字段”的提示，只有输入内容之后才允许提交表单。



图 3.51　提示“请填写此字段”







3.4　新的表单元素

HTML5 新增 3 个表单元素：datalist、keygen 和 output，下面分别进行说明。

3.4.1　datalist——数据列表

datalist 元素用于为输入框提供一个可选的列表，供用户输入匹配或直接选择。如果不想从列表中选择，也可以自行输入内容。

datalist 元素需要与 option 元素配合使用，每一个 option 选项都必须设置 value 属性值。其中 <datalist>标签用于定义列表框，<option>标签用于定义列表项。如果要把 datalist 提供的列表绑定到某输入框上，还需要使用输入框的 list 属性来引用 datalist 元素的 id。

【示例】下面示例演示了 datalist 元素和 list 属性的配合使用。

```
<form action="testform.asp" method="get">
    请输入网址：<input type="url" list="url_list" name="weblink" />
    <datalist id="url_list">
        <option label="新浪" value="http://www.sina.com.cn" />
        <option label="搜狐" value="http://www.sohu.com" />
        <option label="网易" value="http://www.163.com" />
    </datalist>
    <input type="submit" value="提交" />
</form>
```

在 Chrome 浏览器中运行，当用户单击输入框之后，就会弹出一个下拉网址列表，供用户选择，效果如图 3.52 所示。

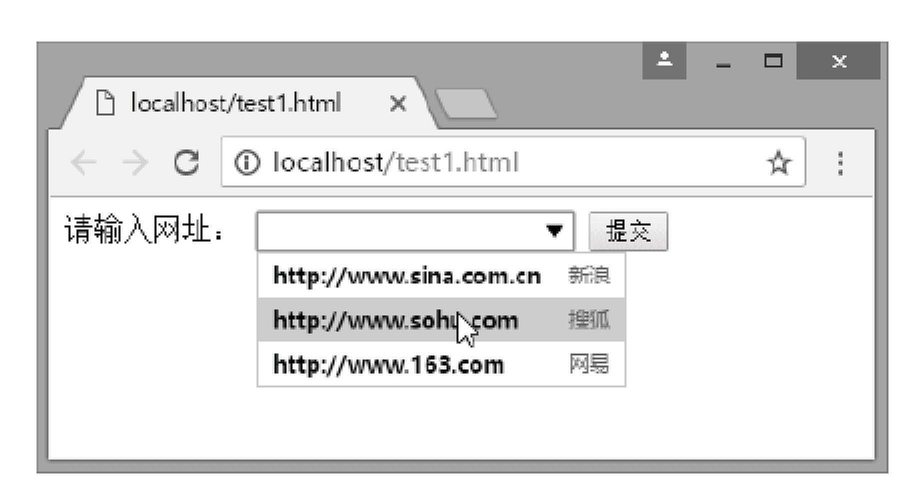


图 3.52　datalist 元素和 list 属性应用

3.4.2　keygen——密钥对生成器



keygen 元素的作用是提供一种验证用户的可靠方法。

作为密钥对生成器，当提交表单时，keygen 元素会生成两个键：私钥和公钥。私钥存储于客户端；公钥被发送到服务器，公钥可用于之后验证用户的客户端证书。

目前，浏览器对该元素的支持不是很理想。

【示例】下面是 keygen 属性的一个应用示例。

```
<form action="/testform.asp" method="get">
    请输入用户名: <input type="text" name="usr_name" /><br>
    请选择加密强度: <keygen name="security" /><br>
```



```
        <input type="submit" value="提交" />
    </form>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.53 所示。在“请选择加密强度”右侧的 keygen 元素中可以选择一种密钥强度，有 2048（高强度）和 1024（中等强度）两种，在 Firefox 浏览器也提供两种选项，如图 3.54 所示。



Note



图 3.53　Chrome 浏览器提供的密钥等级



图 3.54　Firefox 浏览器提供的密钥等级

视频讲解

3.4.3　output——输出结果

output 元素用于在浏览器中显示计算结果或脚本输出，其语法如下。

```
    <output name="">Text</output>
```

【示例】下面是 output 元素的一个应用示例。该示例计算用户输入的两个数字的乘积。

```
    <script type="text/javascript">
     function multi(){
        a=parseInt(prompt("请输入第 1 个数字。",0));
        b=parseInt(prompt("请输入第 2 个数字。",0));
        document.forms["form"]["result"].value=a*b;
     }
    </script>

    <body onload="multi()">
    <form action="testform.asp" method="get" name="form">
        两数的乘积为: <output name="result"></output>
    </form>
    </body>
```

以上代码在 Chrome 浏览器中的运行结果如图 3.55 和图 3.56 所示。当页面载入时，会首先提示“请输入第 1 个数字”，输入并单击“确定”按钮后再根据提示输入第 2 个数字。再次单击“确定”按钮后，显示计算结果，如图 3.57 所示。

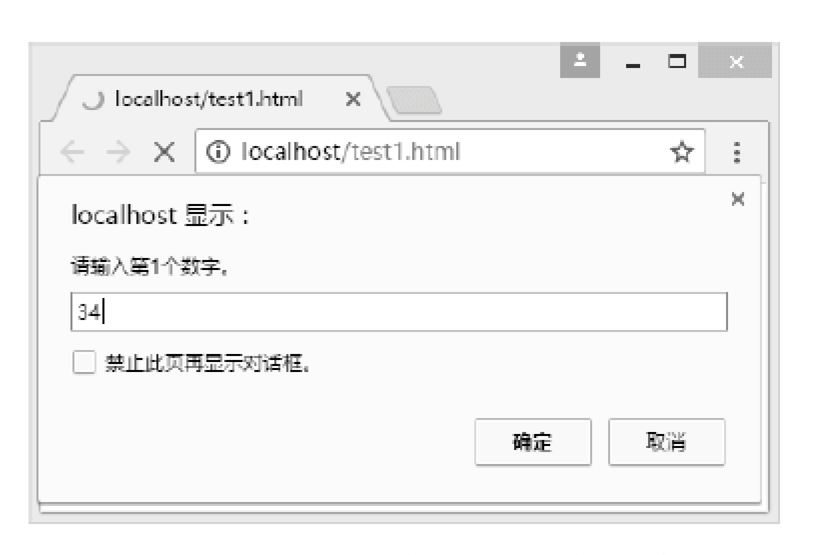


图 3.55　提示输入第 1 个数字

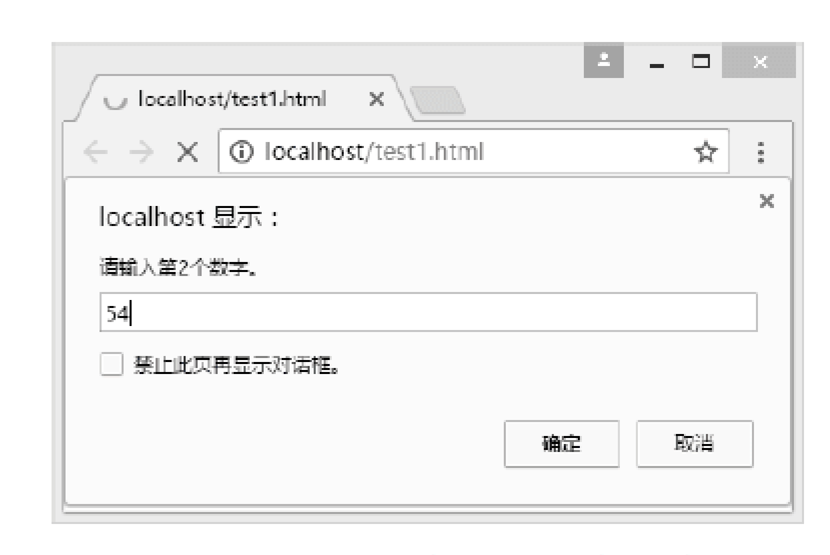


图 3.56　提示输入第 2 个数字



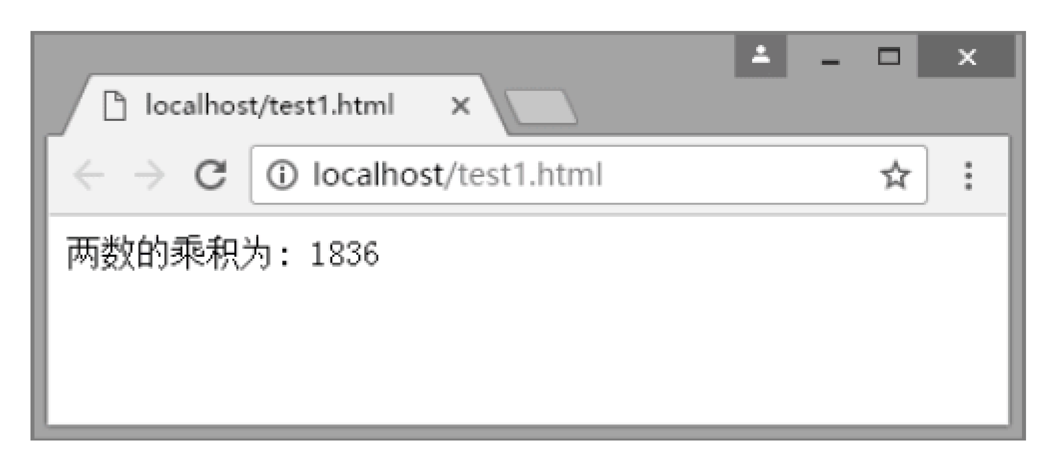
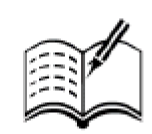



图 3.57　显示计算结果



Note

3.5　新的 form 属性

HTML5 为 form 元素新增了两个属性：autocomplete 和 novalidate，下面分别进行说明。

3.5.1　autocomplete——自动完成



视频讲解

autocomplete 属性用于规定 form 中所有元素都拥有自动完成功能。该属性在介绍 input 属性时已经介绍过，用法与之相同。

但是当 autocomplete 属性用于整个 form 时，所有从属于该 form 的控件都具备自动完成功能。如果要关闭部分控件的自动完成功能，则需要单独设置 autocomplete="off"，具体示例可参考 3.3.1 节 autocomplete 属性的介绍。

3.5.2　novalidate——禁止验证



视频讲解

novalidate 属性规定在提交表单时不应该验证 form 或 input 域。适用于<form>标签，以及 text、search、url、telephone、email、password、date pickers、range 和 color 类型的<input>标签。

【示例 1】下面示例使用 novalidate 属性取消了整个表单的验证。

```
<form action="testform.asp" method="get" novalidate>
    请输入电子邮件地址: <input type="email" name="user_email" />
    <input type="submit" value="提交" />
</form>
```

【补充】

HTML5 为 form、input、select 和 textarea 元素定义了一个 checkValidity()方法。调用该方法，可以显式地对表单内所有元素内容或单个元素内容进行有效性验证。checkValidity()方法将返回布尔值，以提示是否通过验证。

【示例 2】下面示例使用 checkValidity()方法，主动验证用户输入的 Email 地址是否有效。

```
<script>
function check(){
    var email = document.getElementById("email");
    if(email.value==""){
        alert("请输入 Email 地址");
        return false;
    }
    else if(!email.checkValidity()){
```



```
                alert("请输入正确的 Email 地址");
                return false;
            }
            else
                alert("您输入的 Email 地址有效");
        }
    </script>
    <form id=testform onsubmit="return check();" novalidate>
        <label for=email>Email</label>
        <input name=email id=email type=email /><br/>
        <input type=submit>
    </form>
```



Note

提示： 在 HTML5 中，form 和 input 元素都有一个 validity 属性，该属性返回一个 ValidityState 对象。该对象具有很多属性，其中最简单、最重要的属性为 valid 属性，它表示表单内所有元素内容是否有效或单个 input 元素内容是否有效。

3.6 案 例 实 战

下面通过两个案例练习 HTML5 表单在页面中的应用。

视频讲解

3.6.1 设计 HTML5 注册表单

本节示例将利用 HTML5 新的表单功能，设计一个简单的用户注册的界面，注册项目包括用户名、密码、出生日期、国籍、保密问题等内容。本例在 Opera 浏览器中的运行效果如图 3.58 所示。



图 3.58　设计 HTML5 注册表单



线上阅读

具体代码解析请扫码学习。





视 频 讲 解



Note

3.6.2　设计 HTML5 验证表单

本节示例将利用 HTML5 表单内建校验机制，设计一个表单验证页面，效果如图 3.59 所示。

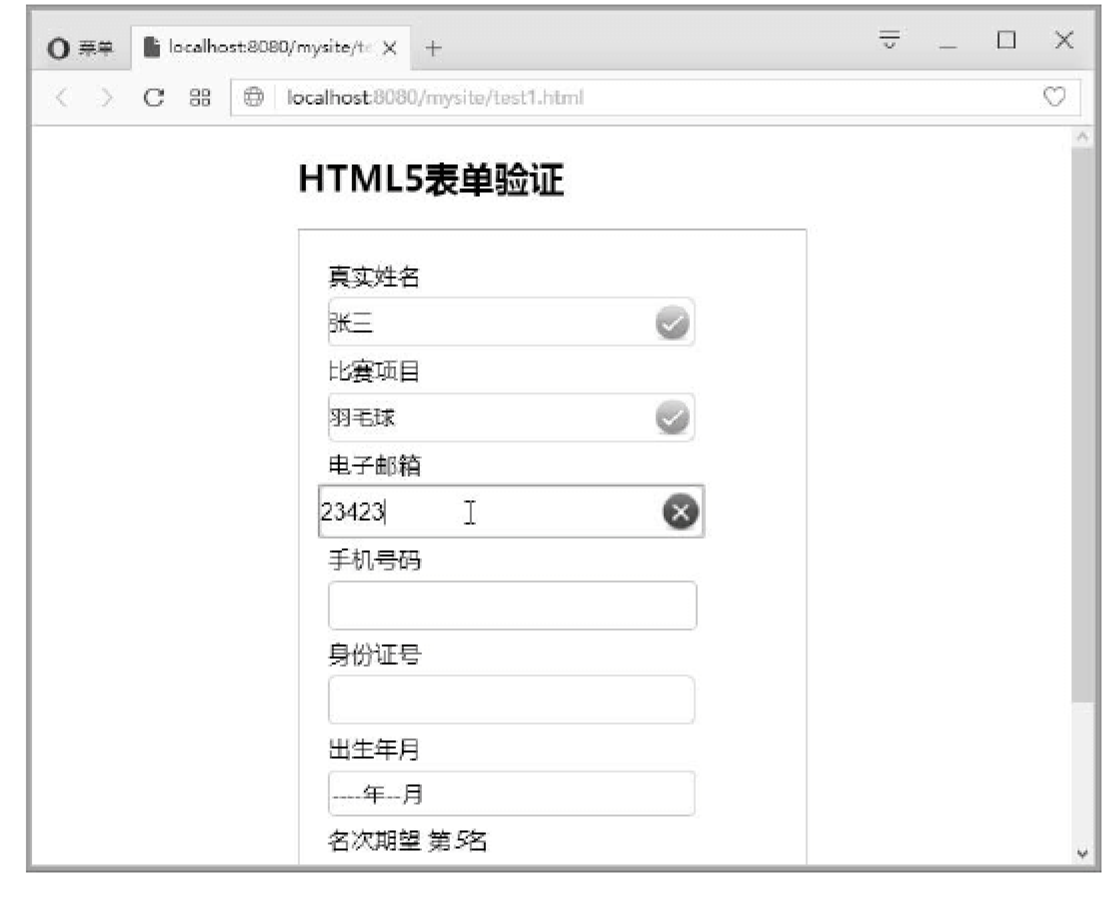


图 3.59　设计 HTML5 验证表单



线 上 阅 读

具体操作步骤请扫码学习。

3.7　在 线 练 习

1. 练习表单结构设计和基本行为的控制。



在 线 练 习

2. 练习使用 CSS3 设计各种网页表单样式，培养网页控件的设计能力。



在 线 练 习



HTML5 绘图和动画

HTML5 新增<canvas>标签，并提供了一套 Canvas API，允许用户通过使用 JavaScript 脚本在<canvas>标签标识的画布上绘制图形、创建动画，甚至可以进行实时视频处理或渲染。本章将重点介绍 Canvas API 的基本用法，帮助用户在网页中绘制漂亮的图形，创造丰富多彩、赏心悦目的 Web 动画。

【学习重点】

- 使用 canvas 元素。
- 绘制图形。
- 设置图形样式。
- 灵活使用 Canvas API 设计网页动画。





Note

4.1 使用 canvas

在 HTML5 文档中，使用<canvas>标签可以在网页中创建一块画布，用法如下所示：

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

该标签包含三个属性：

☑ id：用来标识画布，以方便 JavaScript 脚本对其引用。

☑ width：设置 canvas 的宽度。

☑ height：设置 canvas 的高度。

在默认情况下，canvas 创建的画布大小为宽 300 像素、高 150 像素，可以使用 width 和 height 属性自定义其宽度和高度。

注意，与<img>不同，<canvas>需要结束标签</canvas>。如果结束标签不存在，则文档的其余部分会被认为是替代内容，将不会显示出来。

【示例 1】可以使用 CSS 控制 canvas 的外观。例如，在下面示例中使用 style 属性为 canvas 元素添加一个实心的边框，在浏览器中的预览效果如图 4.1 所示。

```
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>
```

使用 JavaScript 可以在 canvas 画布内绘画，或设计动画。具体步骤如下：

【操作步骤】

第 1 步，在 HTML5 页面中添加<canvas>标签，设置 canvas 的 id 属性值以便 JavaScript 调用。

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

第 2 步，在 JavaScript 脚本中使用 document.getElementById()方法，根据 canvas 元素的 id 获取对 canvas 的引用。

```
var c=document.getElementById("myCanvas");
```

第 3 步，通过 canvas 元素的 getContext()方法获取画布上下文（context），创建 context 对象，以获取允许进行绘制的 2D 环境。

```
var context=c.getContext("2d");
```

getContext("2d")方法返回一个画布渲染上下文对象，使用该对象可以在 canvas 元素中绘制图形，参数“2d”表示二维绘图。

第 4 步，使用 JavaScript 进行绘制。例如，使用以下代码可以绘制一个位于画布中央的矩形。

```
context.fillStyle="#FF00FF";
context.fillRect(50,25,100,50);
```

这两行代码中，fillStyle 属性定义将要绘制的矩形的填充颜色为粉红色，fillRect()方法指定了要绘制的矩形的位置和尺寸。图形的位置由前面的 canvas 坐标值决定，尺寸由后面的宽度和高度值决定。在本例中，坐标值为（50,25），尺寸为宽 100 像素、高 50 像素，根据这些数值，粉红色矩形将出现在画面的中央。

【示例 2】下面给出完整的示例代码。



```
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>
<script>
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.fillStyle="#FF00FF";
context.fillRect(50,25,100,50);
</script>
```

以上代码在浏览器中的预览效果如图 4.2 所示。在画布周围加了边框是为了更能清楚地看到中间矩形位于画布的什么位置。



图 4.1　为 canvas 元素添加实心边框

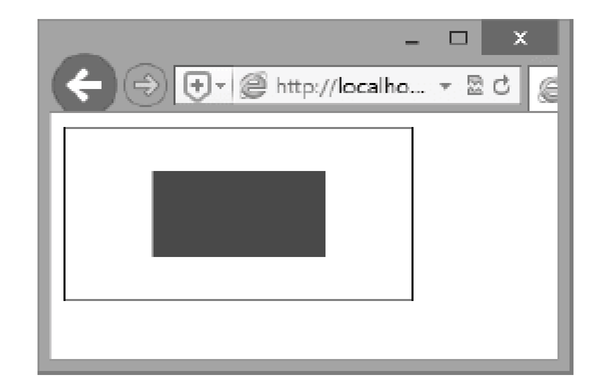


图 4.2　使用 canvas 绘制图形

fillRect(50,25,100,50)方法用来绘制矩形图形，它的前两个参数用于指定绘制图形的 x 轴和 y 轴坐标，后面两个参数设置绘制矩形的宽度和高度。

在 canvas 中，坐标原点（0,0）位于 canvas 画布的左上角，x 轴水平向右延伸，y 轴垂直向下延伸，所有元素的位置都相对于原点进行定位，如图 4.3 所示。

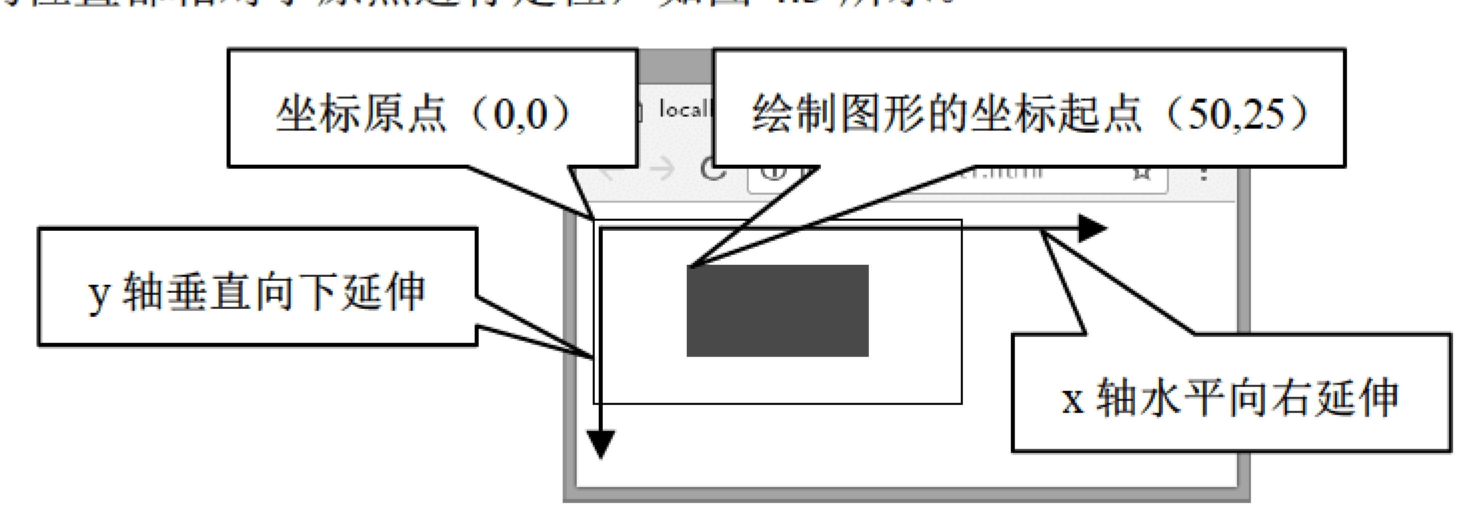


图 4.3　canvas 默认坐标点

目前，IE 9+、Firefox、Opera、Chrome 和 Safari 版本浏览器均支持 canvas 元素及其属性和方法。

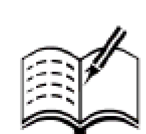
老版本浏览器可能不支持 canvas 元素，因此在特定用户群中，需要为这些浏览器提供替代内容。只需要在<canvas>标签内嵌入替代内容，不支持 canvas 的浏览器会忽略 canvas 元素，而显示替代内容；支持 canvas 的浏览器则会正常渲染 canvas，而忽略替代内容。例如：

```
<canvas id="stockGraph" width="150" height="150">当前浏览器暂不支持 canvas </canvas>
<canvas id="clock" width="150" height="150">
    <img src="images/clock.png" width="150" height="150" alt=""/>
</canvas>
```

注意：canvas 元素可以实现绘图功能，也可以设计动画演示，但是如果 HTML 页面中有比 canvas 元素更合适的元素存在，则建议不要使用 canvas 元素。例如，用 canvas 元素来渲染 HTML 页面的标题样式标签便不太合适。



4.2　绘 制 图 形





本节将介绍一些基本图形的绘制，包括矩形、直线、圆形、曲线等形状或路径。

4.2.1　矩形

canvas 仅支持一种原生的图形绘制：矩形。绘制其他图形都至少需要生成一条路径。不过，拥有众多路径生成的方法，绘制复杂图形也很轻松。

canvas 提供了三种方法绘制矩形：

- ☑　fillRect(x, y, width, height)：绘制一个填充的矩形。
- ☑　strokeRect(x, y, width, height)：绘制一个矩形的边框。
- ☑　clearRect(x, y, width, height)：清除指定矩形区域，让清除部分完全透明。

参数说明如下：

- ☑　x：矩形左上角的 x 坐标。
- ☑　y：矩形左上角的 y 坐标。
- ☑　width：矩形的宽度，以像素为单位。
- ☑　height：矩形的高度，以像素为单位。

【示例】下面示例分别使用上述 3 种方法绘制了 3 个嵌套的矩形，预览效果如图 4.4 所示。

```
<canvas id="canvas" width="300" height="200" style="border:solid 1px #999;"></canvas>
<script>
draw();
function draw() {
    var canvas = document.getElementById('canvas');
    if (canvas.getContext) {
        var ctx = canvas.getContext('2d');
        ctx.fillRect(25,25,100,100);
        ctx.clearRect(45,45,60,60);
        ctx.strokeRect(50,50,50,50);
    }
}
</script>
```

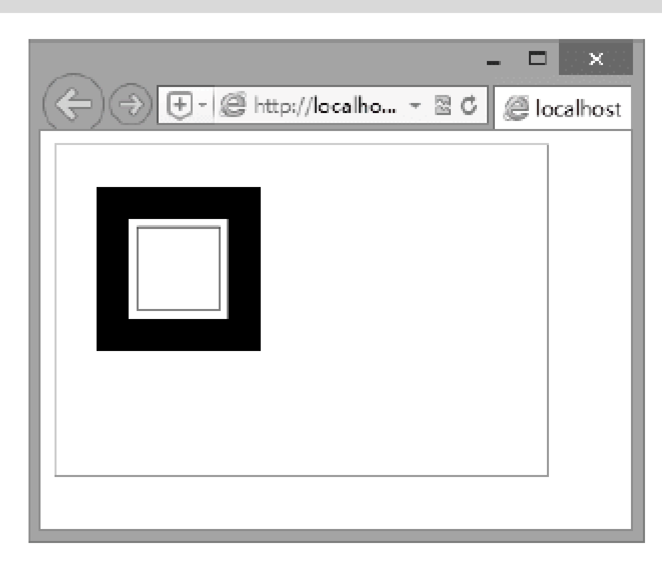


图 4.4　绘制矩形

在上面代码中，fillRect()方法绘制了一个边长为 100px 的黑色正方形。clearRect()方法从正方形的中心开始擦除了一个 60px×60px 的正方形，接着 strokeRect()在清除区域内生成一个 50px×50px 的正方形边框。



视频讲解

Note

提示：不同于路径函数，以上 3 个函数绘制之后，会马上显现在 canvas 上，即时生效。

4.2.2 路径

图形的基本元素是路径。路径是通过不同颜色和宽度的线段或曲线相连形成的不同形状的点的集合。一个路径，甚至一个子路径，都是闭合的。使用路径绘制图形的步骤如下：

第 1 步，创建路径起始点。

第 2 步，使用画图命令绘制路径。

第 3 步，把路径封闭。

第 4 步，生成路径之后，可以通过描边或填充路径区域来渲染图形。

需要调用的方法说明如下：

- ☑ beginPath()：开始路径。新建一条路径，生成之后，图形绘制命令被指向到路径上生成路径。
- ☑ closePath()：闭合路径。闭合路径之后图形绘制命令又重新指向到上下文中。
- ☑ stroke()：描边路径。通过线条来绘制图形轮廓。
- ☑ fill()：填充路径。通过填充路径的内容区域生成实心的图形。

提示：生成路径的第一步是调用 beginPath()方法。每次调用这个方法之后，表示开始重新绘制新的图形。闭合路径 closePath()不是必需的。当调用 fill()方法时，所有没有闭合的形状都会自动闭合，所以不需要调用 closePath()方法，但是调用 stroke()时不会自动闭合。

【示例 1】下面示例绘制一个三角形，效果如图 4.5 所示。代码仅提供绘图函数 draw()，完整代码可以参考 4.2.1 节示例，后面各节示例类似。

```
function draw() {
    var canvas = document.getElementById('canvas');
    if (canvas.getContext){
        var ctx = canvas.getContext('2d');
        ctx.beginPath();
        ctx.moveTo(75,50);
        ctx.lineTo(100,75);
        ctx.lineTo(100,25);
        ctx.fill();
    }
}
```

使用 moveTo(x, y)方法可以将笔触移动到指定的坐标 x 和 y 上。当初始化 canvas，或者调用 beginPath()方法后，通常会使用 moveTo()方法重新设置起点。

【示例 2】用户可以使用 moveTo()方法绘制一些不连续的路径。下面示例绘制一个笑脸图形，效果如图 4.6 所示。

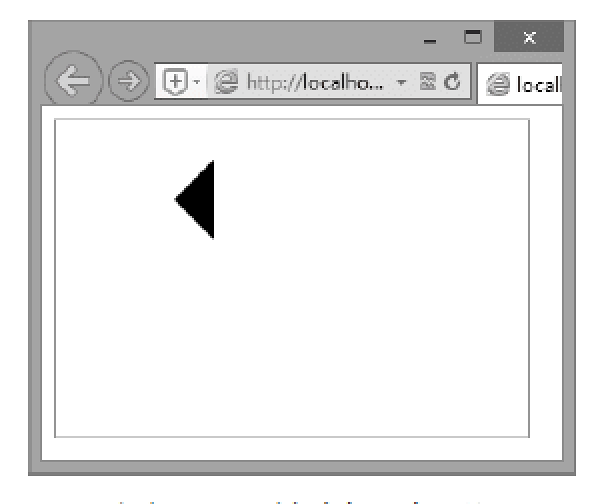


图 4.5 绘制三角形

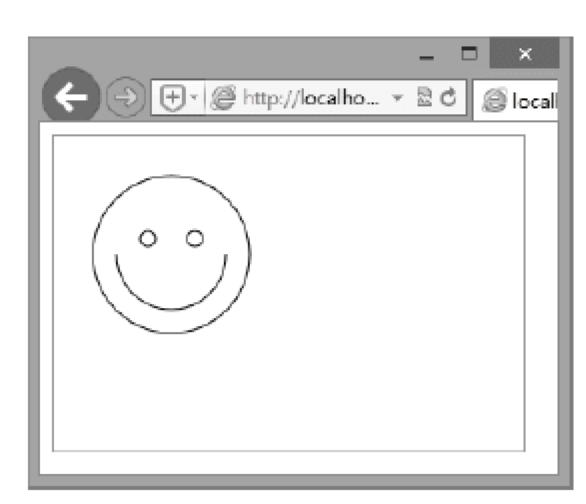
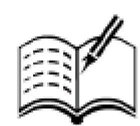


图 4.6 绘制笑脸






```
function draw() {
        var canvas = document.getElementById('canvas');
        if (canvas.getContext){
                var ctx = canvas.getContext('2d');
                ctx.beginPath();
                ctx.arc(75,75,50,0,Math.PI*2,true);          //绘制
                ctx.moveTo(110,75);
                ctx.arc(75,75,35,0,Math.PI,false);           //口（顺时针）
                ctx.moveTo(65,65);
                ctx.arc(60,65,5,0,Math.PI*2,true);           //左眼
                ctx.moveTo(95,65);
                ctx.arc(90,65,5,0,Math.PI*2,true);           //右眼
                ctx.stroke();
        }
}
```

上面代码中使用到 arc()方法，调用它可以绘制圆形，在后面小节中将详细说明。

4.2.3　直线



使用 lineTo()方法可以绘制直线，用法如下所示：

```
lineTo(x,y)
```

参数 x 和 y 分别表示终点位置的 x 坐标和 y 坐标。lineTo(x, y)将绘制一条从当前位置到指定(x, y)位置的直线。

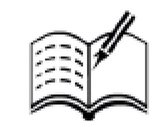
【示例】下面示例将绘制两个三角形，一个是填充的，另一个是描边的，效果如图 4.7 所示。

```
function draw() {
        var canvas = document.getElementById('canvas');
        if (canvas.getContext){
                var ctx = canvas.getContext('2d');
                //填充三角形
                ctx.beginPath();
                ctx.moveTo(25,25);
                ctx.lineTo(105,25);
                ctx.lineTo(25,105);
                ctx.fill();
                //描边三角形
                ctx.beginPath();
                ctx.moveTo(125,125);
                ctx.lineTo(125,45);
                ctx.lineTo(45,125);
                ctx.closePath();
                ctx.stroke();
        }
}
```

在上面示例代码中，从调用 beginPath()方法准备绘制一个新的形状路径开始，使用 moveTo()方法移动到目标位，两条线段绘制后构成三角形的两条边。当路径使用填充（filled）时，路径自动闭合；



而使用描边（stroked）则不会闭合路径。如果没有添加闭合路径 closePath()到描边三角形中，则只绘制了两条线段，并不是一个完整的三角形。



Note

图 4.7　绘制三角形



视频讲解

4.2.4　圆弧

使用 arc()方法可以绘制弧或者圆，用法如下所示：

```
context.arc(x, y, r, sAngle, eAngle, counterclockwise);
```

参数说明如下：

- ☑ x：圆心的 x 坐标。
- ☑ y：圆心的 y 坐标。
- ☑ r：圆的半径。
- ☑ sAngle：起始角，以弧度计。（提示：弧的圆形的三点钟位置是 0°。）
- ☑ eAngle：结束角，以弧度计。
- ☑ counterclockwise：可选参数，定义绘图方向。false 为顺时针，为默认值，true 为逆时针。

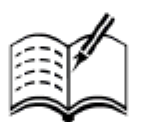
如果使用 arc()创建圆，可以把起始角设置为 0，结束角设置为 2*Math.PI。

【示例 1】下面示例绘制了 12 个不同的角度以及填充的圆弧。主要使用两个 for 循环，生成圆弧的行列（x,y）坐标。每一段圆弧的开始都调用 beginPath()方法。代码中，每个圆弧的参数都是可变的，（x,y）坐标是可变的，半径（radius）和开始角度（startAngle）都是固定的。结束角度（endAngle）在第一列开始时是 180°（半圆），然后每列增加 90°。最后一列形成一个完整的圆。效果如图 4.8 所示。

```
function draw() {
    var canvas = document.getElementById('canvas');
    if (canvas.getContext){
        var ctx = canvas.getContext('2d');
        for(var i=0;i<4;i++){
            for(var j=0;j<3;j++){
                ctx.beginPath();
                var x              = 25+j*50;               // x 坐标值
                var y              = 25+i*50;               // y 坐标值
                var radius         = 20;                    //圆弧半径
                var startAngle     = 0;                     //开始点
                var endAngle       = Math.PI+(Math.PI*j)/2; //结束点
                var anticlockwise  = i%2==0 ? false : true; //顺时针或逆时针
                ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);
                if (i>1){
                    ctx.fill();
                } else {
```



```
                        ctx.stroke();
                    }
                }
            }
        }
    }
```



在上面代码中，“var anticlockwise= i%2==0 ? false : true;”语句作用于第一、三行是顺时针的圆弧，anticlockwise 作用于二、四行为逆时针圆弧。if 语句让一、二行描边圆弧，下面两行填充路径。

使用 arcTo()方法可以绘制曲线，该方法是 lineTo()的曲线版，它能够创建两条切线之间的弧或曲线。用法如下所示：

```
context.arcTo(x1,y1,x2,y2,r);
```

参数说明如下：

☑ x1：弧的起点的 x 坐标。
☑ y1：弧的起点的 y 坐标。
☑ x2：弧的终点的 x 坐标。
☑ y2：弧的终点的 y 坐标。
☑ r：弧的半径。

【示例 2】本例使用 lineTo()和 arcTo()方法绘制直线和曲线，然后连成圆角弧线，效果如图 4.9 所示。

```
function draw() {
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    ctx.beginPath();
    ctx.moveTo(20,20);                    //设置起点
    ctx.lineTo(100,20);                   //绘制水平直线
    ctx.arcTo(150,20,150,70,50);          //绘制曲线
    ctx.lineTo(150,120);                  //绘制垂直直线
    ctx.stroke();                         //开始绘制
}
```

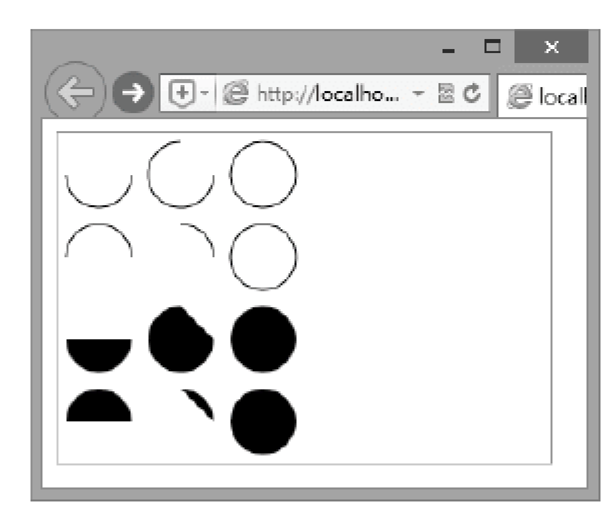


图 4.8　绘制圆和弧

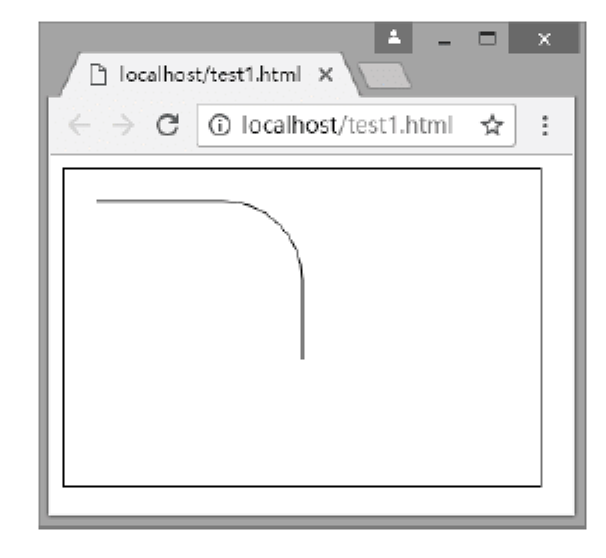


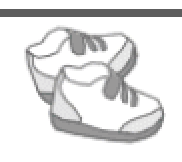
图 4.9　绘制圆角弧线

4.2.5　二次方曲线


视频讲解

使用 quadraticCurveTo()方法可以绘制二次方贝塞尔曲线，用法如下。

```
context.quadraticCurveTo(cpx,cpy,x,y);
```



Note

参数说明如下：

☑ cpx：贝塞尔控制点的 x 坐标。

☑ cpy：贝塞尔控制点的 y 坐标。

☑ x：结束点的 x 坐标。

☑ y：结束点的 y 坐标。

二次方贝塞尔曲线需要两个点。第一个点是用于二次贝塞尔计算中的控制点，第二个点是曲线的结束点。曲线的开始点是当前路径中最后一个点。如果路径不存在，需要使用 beginPath()和 moveTo()方法来定义开始点，演示说明如图 4.10 所示。

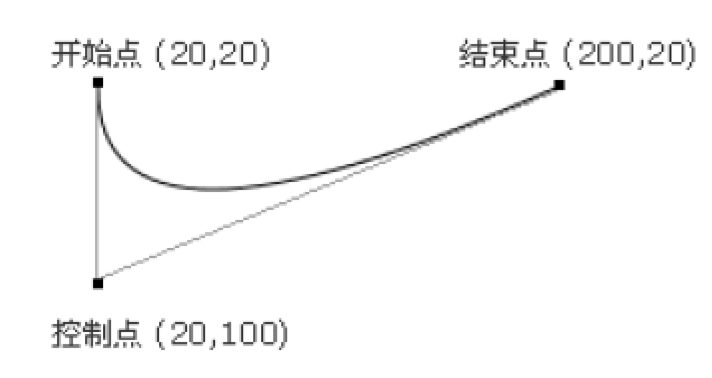


图 4.10　二次方贝塞尔曲线演示示意图

操作步骤如下：

第 1 步，确定开始点，如 moveTo(20,20)。

第 2 步，定义控制点，如 quadraticCurveTo(20,100, x , y)。

第 3 步，定义结束点，如 quadraticCurveTo(20,100,200,20)。

【示例 1】下面示例先绘制一条二次方贝塞尔曲线，再绘制出其控制点和控制线。

```
function draw() {
    var canvas = document.getElementById('canvas');
    var ctx=canvas.getContext("2d");
    //下面开始绘制二次方贝塞尔曲线
    ctx.strokeStyle="dark";
    ctx.beginPath();
    ctx.moveTo(0,200);
    ctx.quadraticCurveTo(75,50,300,200);
    ctx.stroke();
    ctx.globalCompositeOperation="source-over";
    //绘制直线，表示曲线的控制点和控制线，控制点坐标即两直线的交点（75,50）
    ctx.strokeStyle="#ff00ff";
    ctx.beginPath();
    ctx.moveTo(75,50);
    ctx.lineTo(0,200);
    ctx.moveTo(75,50);
    ctx.lineTo(300,200);
    ctx.stroke();
}
```

在浏览器中的运行效果如图 4.11 所示，其中曲线即为二次方贝塞尔曲线，两条直线为控制线，两直线的交点即为曲线的控制点。

【示例 2】下面示例组合直线和二次方曲线，封装了一个圆角矩形函数，使用它可以绘制圆角矩形图形，效果如图 4.12 所示。

具体代码解析请扫码学习。



线 上 阅 读



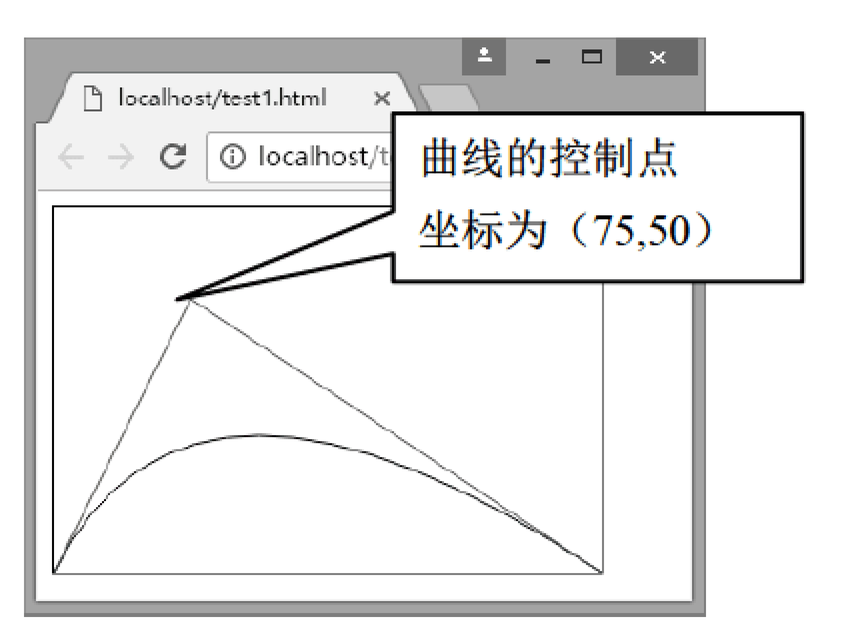


图 4.11　二次方贝塞尔曲线及其控制点

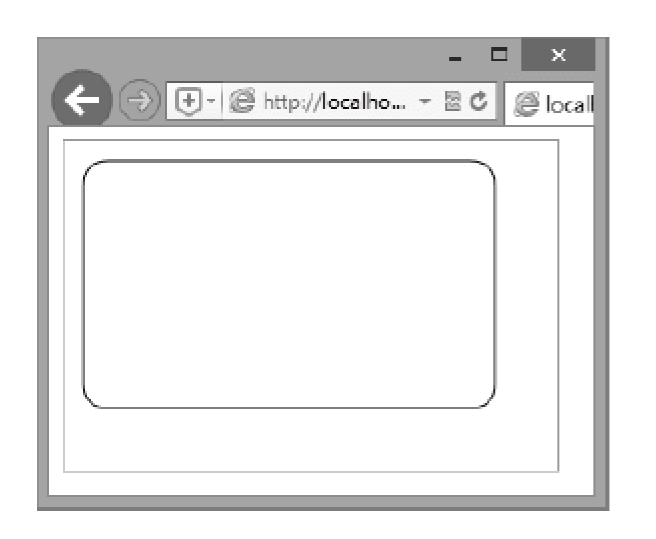
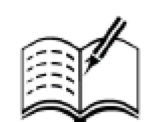


图 4.12　绘制圆角矩形





4.2.6　三次方曲线

使用 bezierCurveTo()方法可以绘制三次方贝塞尔曲线，用法如下。

```
context.bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y);
```

参数说明如下：

☑　cp1x：第一个贝塞尔控制点的 x 坐标。
☑　cp1y：第一个贝塞尔控制点的 y 坐标。
☑　cp2x：第二个贝塞尔控制点的 x 坐标。
☑　cp2y：第二个贝塞尔控制点的 y 坐标。
☑　x：结束点的 x 坐标。
☑　y：结束点的 y 坐标。

三次方贝塞尔曲线需要三个点，前两个点是用于三次贝塞尔计算中的控制点，第三个点是曲线的结束点。曲线的开始点是当前路径中最后一个点，如果路径不存在，需要使用 beginPath()和 moveTo()方法来定义开始点，演示说明如图 4.13 所示。

操作步骤如下：

第 1 步，确定开始点，如 moveTo(20,20)。

第 2 步，定义第一个控制点，如 bezierCurveTo(20, 100, cp2x, cp2y, x, y)。

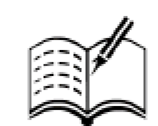
第 3 步，定义第二个控制点，如 bezierCurveTo(20,100,200,100, x, y)。

第 4 步，定义结束点，如 bezierCurveTo(20,100,200,100,200,20)。

【示例】下面示例绘制了一条三次方贝塞尔曲线，还绘制出了两个控制点和两条控制线。

```
function draw() {
    var canvas = document.getElementById('canvas');
    var ctx=canvas.getContext("2d");
    //下面开始绘制三次方贝塞尔曲线
    ctx.strokeStyle="dark";
    ctx.beginPath();
    ctx.moveTo(0,200);
    ctx.bezierCurveTo(25,50,75,50,300,200);
    ctx.stroke();
    ctx.globalCompositeOperation="source-over";
    //下面绘制直线，用于表示上面曲线的控制点和控制线，控制点坐标为（25,50）和（75,50）
    ctx.strokeStyle="#ff00ff";
```





Note

```
        ctx.beginPath();
        ctx.moveTo(25,50);
        ctx.lineTo(0,200);
        ctx.moveTo(75,50);
        ctx.lineTo(300,200);
        ctx.stroke();
}
```

在浏览器中的预览效果如图 4.14 所示，其中曲线即为三次方贝塞尔曲线，两条直线为控制线，两直线上方的端点即为曲线的控制点。

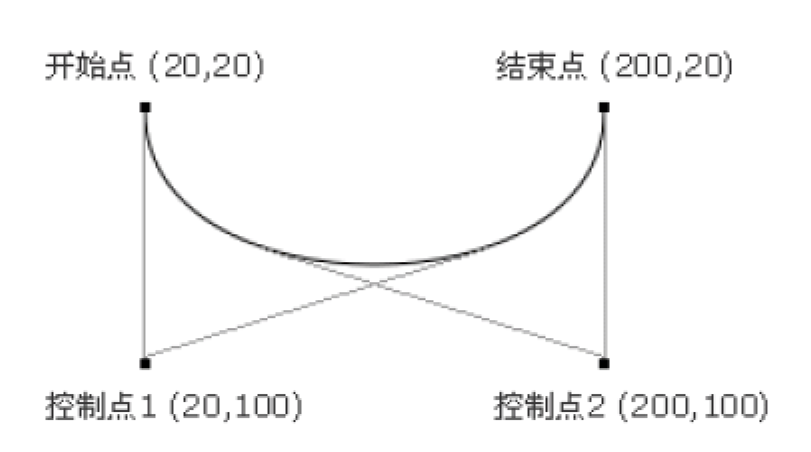


图 4.13　三次方贝塞尔曲线演示示意图

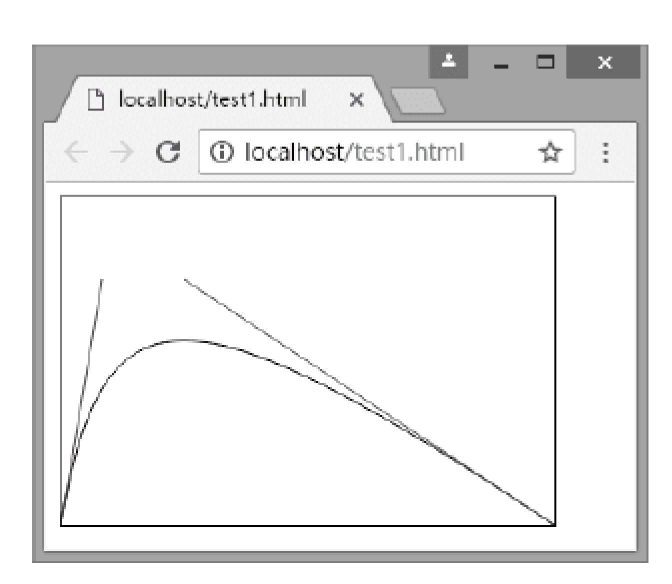


图 4.14　三次方贝塞尔曲线

4.3　定义样式和颜色

canvas 支持很多颜色和样式选项，如线型、渐变、图案、透明度和阴影。本节将介绍样式的设置方法。



视 频 讲 解

4.3.1　颜色

使用 fillStyle 和 strokeStyle 属性可以给图形上色。其中，fillStyle 设置图形的填充颜色，strokeStyle 设置图形轮廓的颜色。

颜色值可以是表示 CSS 颜色值的字符串，也可以是渐变对象或者图案对象（参考下面小节介绍）。默认情况下，线条和填充颜色都是黑色，CSS 颜色值为#000000。

一旦设置了 strokeStyle 或 fillStyle 的值，那么这个新值就会成为新绘制的图形的默认值。如果要给每个图形定义不同的颜色，就需要重新设置 fillStyle 或 strokeStyle 的值。

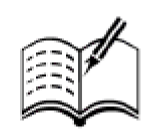
【示例 1】本例使用嵌套 for 循环绘制方格阵列，每个方格填充不同色，效果如图 4.15 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    for (var i=0;i<6;i++){
        for (var j=0;j<6;j++){
            ctx.fillStyle = 'rgb(' + Math.floor(255-42.5*i) + ',' + Math.floor(255-42.5*j) + ',0)';
            ctx.fillRect(j*25,i*25,25,25);
        }
    }
}
```



在嵌套 for 结构中，使用变量 i 和 j 为每个方格产生唯一的 RGB 色彩值，其中仅修改红色和绿色通道的值，而保持蓝色通道的值不变。可以通过修改这些颜色通道的值来产生各种各样的色板。通过增加渐变的频率，可以绘制出类似 Photoshop 调色板的效果。

【示例 2】下面示例与示例 1 有点类似，但使用 strokeStyle 属性，画的不是方格，而是用 arc()方法画圆，效果如图 4.16 所示。




```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    for (var i=0;i<6;i++){
        for (var j=0;j<6;j++){
            ctx.strokeStyle = 'rgb(0,' + Math.floor(255-42.5*i) + ',' + Math.floor(255-42.5*j) + ')';
            ctx.beginPath();
            ctx.arc(12.5+j*25,12.5+i*25,10,0,Math.PI*2,true);
            ctx.stroke();
        }
    }
}
```

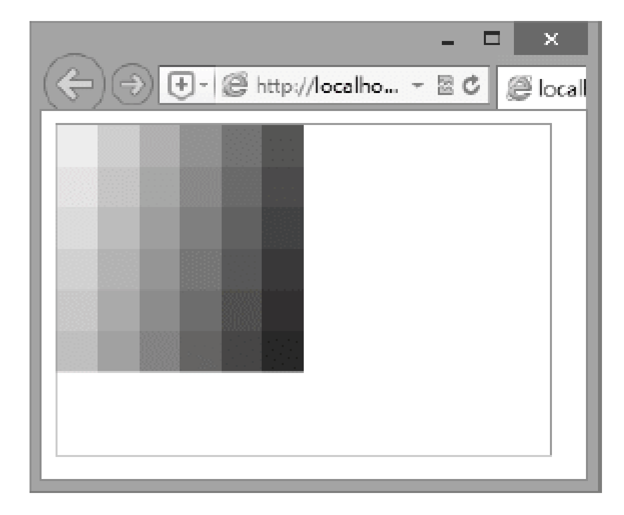


图 4.15　绘制渐变色块

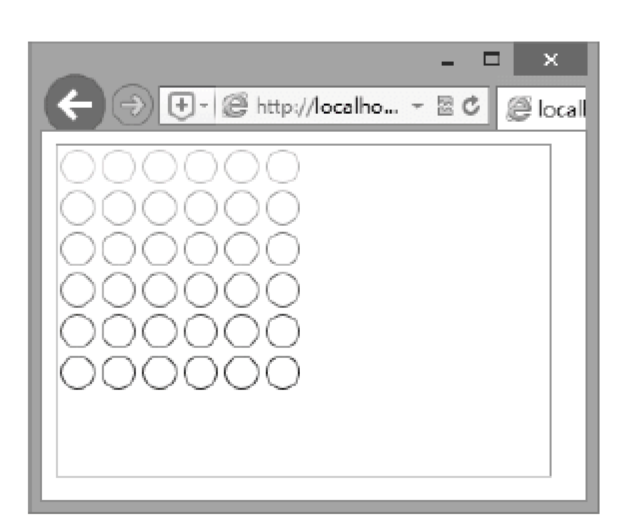


图 4.16　绘制渐变圆圈

4.3.2　不透明度



使用 globalAlpha 全局属性可以设置绘制图形的不透明度，另外也可以通过色彩的不透明度参数来为图形设置不透明度，这种方法相对于使用 globalAlpha 属性来说，会更灵活些。

使用 rgba()方法可以设置具有不透明度的颜色，用法如下。

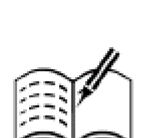
```
rgba(R,G,B,A)
```

其中 R、G、B 将颜色的红色、绿色和蓝色成分指定为 0～255 的十进制整数，A 把 alpha（不透明）成分指定为 0.0 和 1.0 之间的一个浮点数值，0.0 为完全透明，1.0 为完全不透明。例如，可以用"rgba(255,0,0,0.5)"表示半透明的完全红色。

【示例 1】下面示例使用四色格作为背景，设置 globalAlpha 为 0.2 后，在上面画一系列半径递增的半透明圆，最终结果是一个径向渐变效果，如图 4.17 所示。圆叠加得越多，原先所画的圆的透明度会越低。通过增加循环次数，画更多的圆，背景图的中心部分会完全消失。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    //画背景
    ctx.fillStyle = '#FD0';
    ctx.fillRect(0,0,75,75);
    ctx.fillStyle = '#6C0';
```





Note

```
        ctx.fillRect(75,0,75,75);
        ctx.fillStyle = '#09F';
        ctx.fillRect(0,75,75,75);
        ctx.fillStyle = '#F30';
        ctx.fillRect(75,75,75,75);
        ctx.fillStyle = '#FFF';
        //设置透明度值
        ctx.globalAlpha = 0.2;
        //画半透明圆
        for (var i=0;i<7;i++){
            ctx.beginPath();
            ctx.arc(75,75,10+10*i,0,Math.PI*2,true);
            ctx.fill();
        }
}
```



线上阅读

【示例 2】本例与示例 1 类似，不过不是画圆，而是画矩形。这里还可以看出，rgba()可以分别设置轮廓和填充样式，因而具有更好的可操作性和使用灵活性，效果如图 4.18 所示。

具体代码解析请扫码学习。

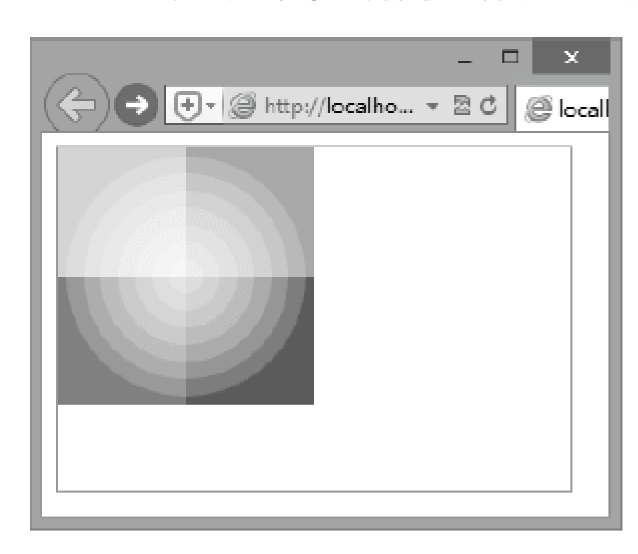


图 4.17　用 globalAlpha 设置不透明度

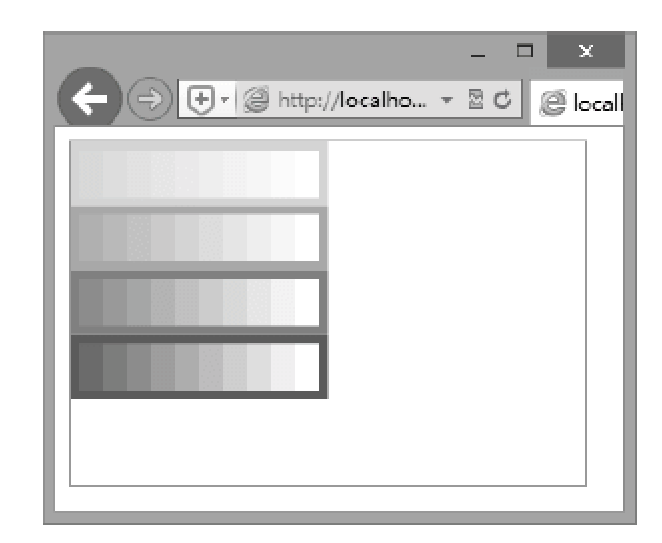


图 4.18　用 rgba()方法设置不透明度



视频讲解

4.3.3　实线

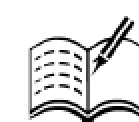
1. 线的粗细

使用 lineWidth 属性可以设置线条的粗细，取值必须为正数，默认为 1.0。

【示例 1】下面示例使用 for 循环绘制了 12 条线宽依次递增的线段，效果如图 4.19 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    for (var i = 0; i < 12; i++){
        ctx.strokeStyle="red";
        ctx.lineWidth = 1+i;
        ctx.beginPath();
        ctx.moveTo(5,5+i*14);
        ctx.lineTo(140,5+i*14);
        ctx.stroke();
    }
}
```





2. 端点样式

lineCap 属性用于设置线段端点的样式，包括三种样式：butt、round 和 square，默认值为 butt。

【示例 2】下面示例绘制了三条蓝色的直线段，并依次设置上述三种属性值，两侧有两条红色的参考线，以方便观察，预览效果如图 4.20 所示。可以看到这三种端点样式从上到下依次为平头、圆头和方头。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var lineCap = ['butt','round','square'];
    //绘制参考线
    ctx.strokeStyle = 'red';
    ctx.beginPath();
    ctx.moveTo(10,10);
    ctx.lineTo(10,150);
    ctx.moveTo(150,10);
    ctx.lineTo(150,150);
    ctx.stroke();
    //绘制直线段
    ctx.strokeStyle = 'blue';
    for (var i=0;i<lineCap.length;i++){
        ctx.lineWidth = 20;
        ctx.lineCap = lineCap[i];
        ctx.beginPath();
        ctx.moveTo(10,30+i*50);
        ctx.lineTo(150,30+i*50);
        ctx.stroke();
    }
}
```



图 4.19　lineWidth 示例



图 4.20　lineCap 示例

3. 连接样式

lineJoin 属性用于设置两条线段连接处的样式，包括三种样式：round、bevel 和 miter，默认值为 miter。

【示例 3】下面示例绘制了三条蓝色的折线，并依次设置上述三种属性值，观察拐角处（即直线段连接处）样式的区别。在浏览器中的预览效果如图 4.21 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var lineJoin = ['round','bevel','miter'];
    ctx.strokeStyle = 'blue';
```

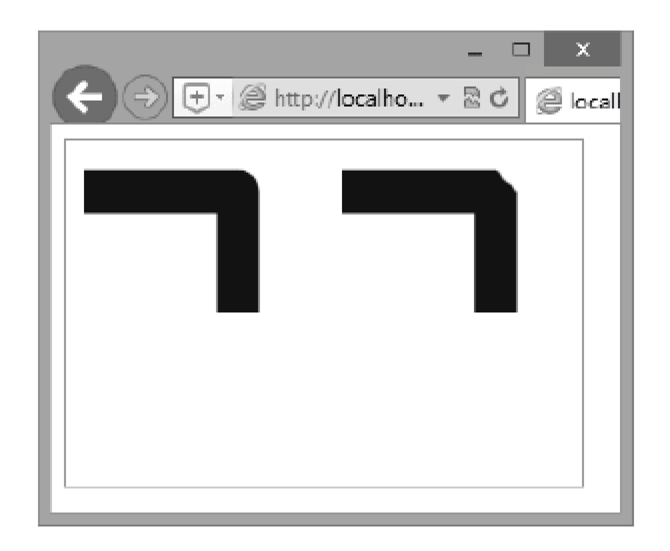
```
        for (var i=0;i<lineJoin.length;i++){
            ctx.lineWidth = 25;
            ctx.lineJoin = lineJoin[i];
            ctx.beginPath();
            ctx.moveTo(10+i*150,30);
            ctx.lineTo(100+i*150,30);
            ctx.lineTo(100+i*150,100);
            ctx.stroke();
        }
    }
```

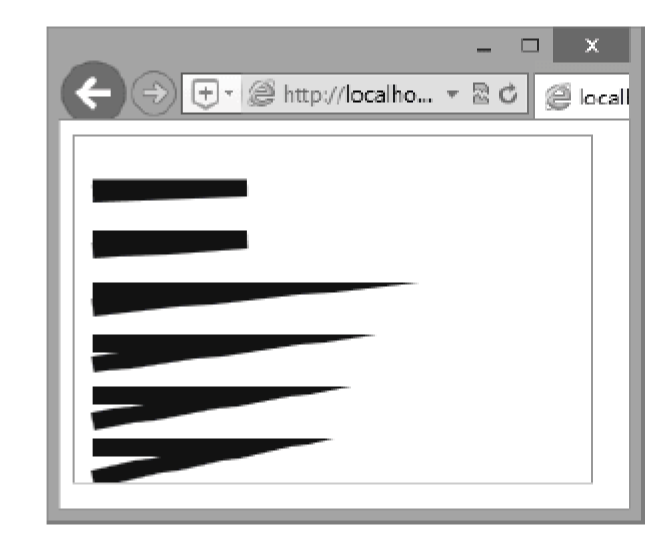
4. 交点方式

miterLimit 属性用于设置两条线段连接处交点的绘制方式，其作用是为斜面的长度设置一个上限，默认为 10，即规定斜面的长度不能超过线条宽度的 10 倍。当斜面的长度达到线条宽度的 10 倍时，就会变为斜角。如果 lineJoin 属性值为 round 或 bevel 时，miterLimit 属性无效。

【示例 4】通过下面示例可以观察当角度和 miterLimit 属性值发生变化时斜面长度的变化。在运行代码之前，也可以将 miterLimit 属性值改为固定值，以观察不同的值产生的结果，效果如图 4.22 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    for (var i=1;i<10;i++){
        ctx.strokeStyle = 'blue';
        ctx.lineWidth = 10;
        ctx.lineJoin = 'miter';
        ctx.miterLimit = i*10;
        ctx.beginPath();
        ctx.moveTo(10,i*30);
        ctx.lineTo(100,i*30);
        ctx.lineTo(10,33*i);
        ctx.stroke();
    }
}
```


图 4.21　lineJoin 示例

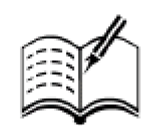

图 4.22　miterLimit 示例

4.3.4　虚线


视频讲解

使用 setLineDash()方法和 lineDashOffset 属性可以定义虚线样式。setLineDash()方法接收一个数组来指定线段与间隙的交替，lineDashOffset 属性设置起始偏移量。





【示例】下面示例绘制一个矩形虚线框，然后使用定时器设计每隔 0.5 秒重绘一次，重绘时改变 lineDashOffset 属性值，从而创建一个行军蚁的效果，效果如图 4.23 所示。

```
var ctx = document.getElementById('canvas').getContext('2d');
var offset = 0;
function draw() {
    ctx.clearRect(0,0, canvas.width, canvas.height);
    ctx.setLineDash([4, 4]);
    ctx.lineDashOffset = -offset;
    ctx.strokeRect(50,50, 200, 100);
}
function march() {
    offset++;
    if (offset > 16) {
        offset = 0;
    }
    draw();
    setTimeout(march, 100);
}
march();
```

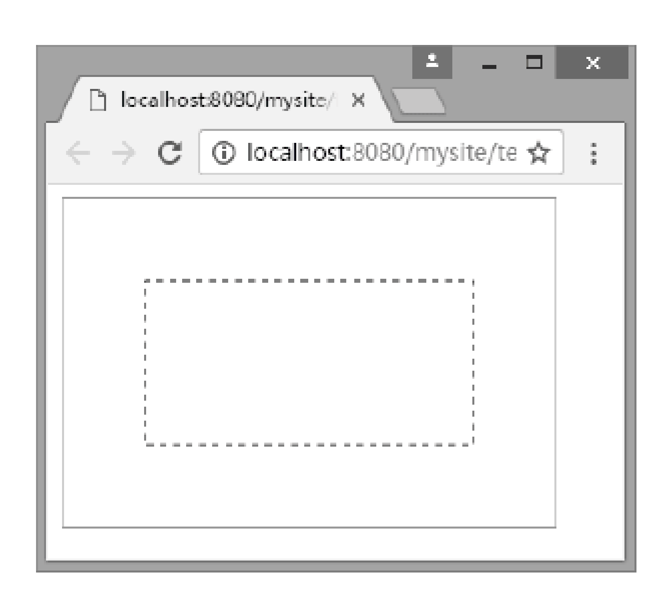


图 4.23　设计动态虚线框

注意：在 IE 浏览器中，从 IE 11 开始才支持 setLineDash()方法和 lineDashOffset 属性。

4.3.5　线性渐变



要绘制线性渐变，首先使用 createLinearGradient()方法创建 canvasGradient 对象，然后使用 addColorStop()方法进行上色。

createLinearGradient()方法的用法如下所示：

```
context.createLinearGradient(x0,y0,x1,y1);
```

参数说明如下：

☑　x0：渐变开始点的 x 坐标。
☑　y0：渐变开始点的 y 坐标。
☑　x1：渐变结束点的 x 坐标。
☑　y1：渐变结束点的 y 坐标。

addColorStop()方法的用法如下所示：

```
gradient.addColorStop(stop,color);
```



参数说明如下：

☑ stop：介于 0.0 与 1.0 的值，表示渐变中开始与结束之间的相对位置。渐变起点的偏移值为 0，终点的偏移值为 1。如果 position 值为 0.5，则表示色标会出现在渐变的正中间。

☑ color：在结束位置显示的 CSS 颜色值。

【示例】下面示例演示如何绘制线性渐变。在本例中共添加了 8 个色标，分别为红、橙、黄、绿、青、蓝、紫、红，预览效果如图 4.24 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var lingrad = ctx.createLinearGradient(0,0,0,200);
    lingrad.addColorStop(0, '#ff0000');
    lingrad.addColorStop(1/7, '#ff9900');
    lingrad.addColorStop(2/7, '#ffff00');
    lingrad.addColorStop(3/7, '#00ff00');
    lingrad.addColorStop(4/7, '#00ffff');
    lingrad.addColorStop(5/7, '#0000ff');
    lingrad.addColorStop(6/7, '#ff00ff');
    lingrad.addColorStop(1, '#ff0000');
    ctx.fillStyle = lingrad;
    ctx.strokeStyle = lingrad;
    ctx.fillRect(0,0,300,200);
}
```

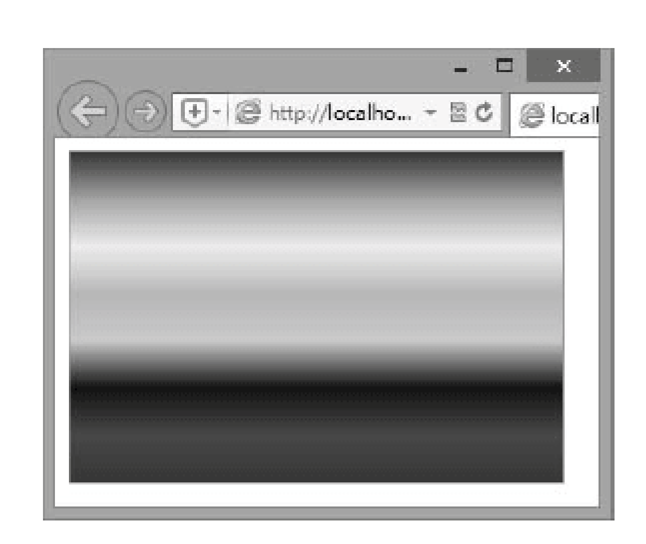


图 4.24　绘制线性渐变

使用 addColorStop()方法可以添加多个色标，色标可以在 0～1 任意位置添加，例如，从 0.3 处开始设置一个蓝色色标，再在 0.5 处设置一个红色色标，则 0～0.3 都会填充为蓝色。0.3～0.5 为蓝色到红色的渐变，0.5～1 则填充为红色。

视频讲解

4.3.6　径向渐变

要绘制径向渐变，首先需要使用 createRadialGradient()方法创建 canvasGradient 对象，然后使用 addColorStop()方法进行上色。

createRadialGradient()方法的用法如下。

```
context.createRadialGradient(x0,y0,r0,x1,y1,r1);
```

参数说明如下：

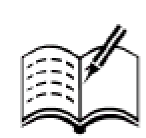
☑ x0：渐变的开始圆的 x 坐标。

☑ y0：渐变的开始圆的 y 坐标。



☑　r0：开始圆的半径。
☑　x1：渐变的结束圆的 x 坐标。
☑　y1：渐变的结束圆的 y 坐标。
☑　r1：结束圆的半径。

【示例】下面示例使用径向渐变在画布中央绘制一个圆球形状，预览效果如图 4.25 所示。



```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    //创建渐变
    var radgrad = ctx.createRadialGradient(150,100,0,150,100,100);
    radgrad.addColorStop(0, '#A7D30C');
    radgrad.addColorStop(0.9, '#019F62');
    radgrad.addColorStop(1, 'rgba(1,159,98,0)');
    //填充渐变色
    ctx.fillStyle = radgrad;
    ctx.fillRect(0,0,300,200);
}
```

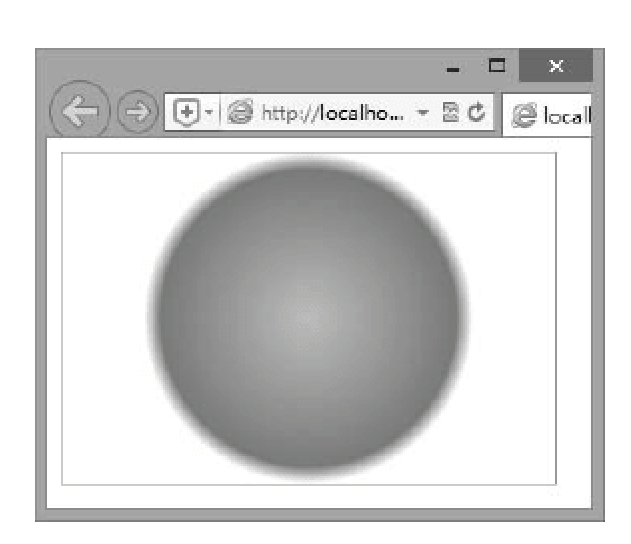


图 4.25　绘制径向渐变

4.3.7　图案



视 频 讲 解

使用 createPattern()方法可以绘制图案效果，用法如下所示。

```
context.createPattern(image,"repeat|repeat-x|repeat-y|no-repeat");
```

参数说明如下：

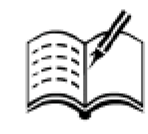
☑　image：规定要使用的图片、画布或视频元素。
☑　repeat：默认值。该模式在水平和垂直方向重复。
☑　repeat-x：该模式只在水平方向重复。
☑　repeat-y：该模式只在垂直方向重复。
☑　no-repeat：该模式只显示一次（不重复）。

创建图案的步骤与创建渐变有些类似，需要先创建出一个 pattern 对象，然后将其赋予 fillStyle 属性或 strokeStyle 属性。

【示例】下面示例以一幅 png 格式的图像作为 image 对象用于创建图案，以平铺方式同时沿 x 轴与 y 轴方向平铺。在浏览器中的预览效果如图 4.26 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    //创建用于图案的新 image 对象
    var img = new Image();
```





Note

```
        img.src = 'images/1.png';
        img.onload = function(){
            //创建图案
            var ptrn = ctx.createPattern(img,'repeat');
            ctx.fillStyle = ptrn;
            ctx.fillRect(0,0,600,600);
        }
}
```

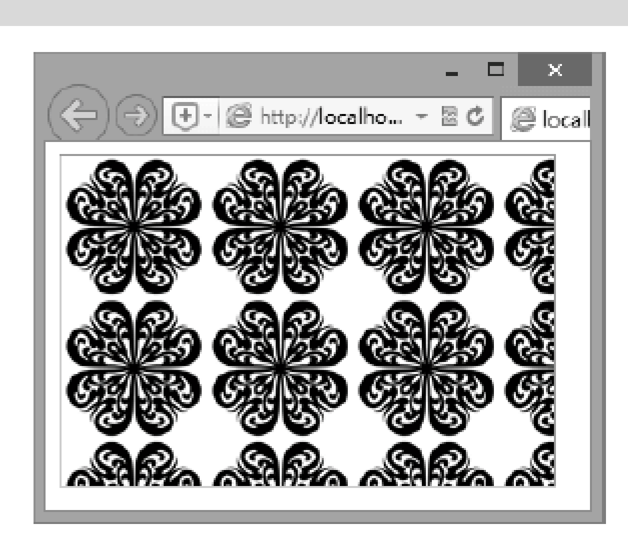


图 4.26　绘制图案



视 频 讲 解

4.3.8　阴影

创建阴影需要 4 个属性，简单说明如下：

☑　shadowColor：设置阴影颜色。

☑　shadowBlur：设置阴影的模糊级别。

☑　shadowOffsetX：设置阴影在 x 轴的偏移距离。

☑　shadowOffsetY：设置阴影在 y 轴的偏移距离。

【示例】下面示例演示创建文字阴影效果，如图 4.27 所示。

```
function draw() {
        var ctx = document.getElementById('canvas').getContext('2d');
        //设置阴影
        ctx.shadowOffsetX = 4;
        ctx.shadowOffsetY = 4;
        ctx.shadowBlur = 4;
        ctx.shadowColor = "rgba(0, 0, 0, 0.5)";
        //绘制文本
        ctx.font = "60px Times New Roman";
        ctx.fillStyle = "Black";
        ctx.fillText("Canvas API", 5, 80);
}
```

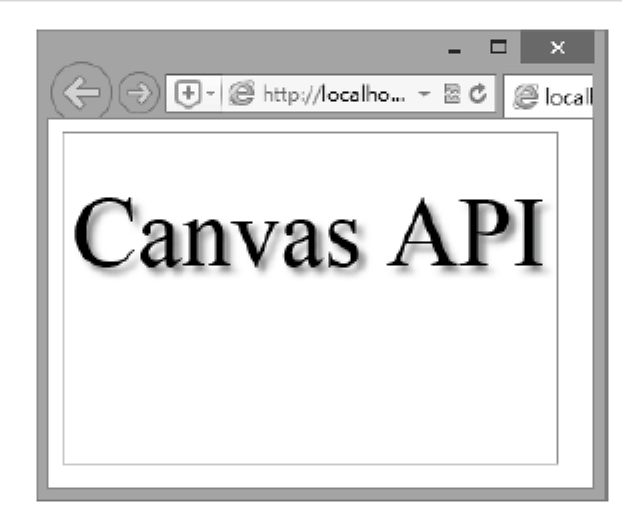


图 4.27　为文字设置阴影效果





视 频 讲 解

Note

4.3.9 填充规则

前面介绍了使用 fill()方法可以填充图形，该方法可以接收两个值，用来定义填充规则。取值说明如下：

☑ "nonzero"：非零环绕数规则，为默认值。

☑ "evenodd"：奇偶规则。

填充规则根据某处在路径的外面或者里面来决定该处是否被填充，这对于路径相交或者路径被嵌套的时候是有用的。

【示例】下面示例使用 evenodd 规则填充图形，则效果如图 4.28 所示，默认填充效果如图 4.29 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.beginPath();
    ctx.arc(50, 50, 30, 0, Math.PI*2, true);
    ctx.arc(50, 50, 15, 0, Math.PI*2, true);
    ctx.fill("evenodd");
}
```

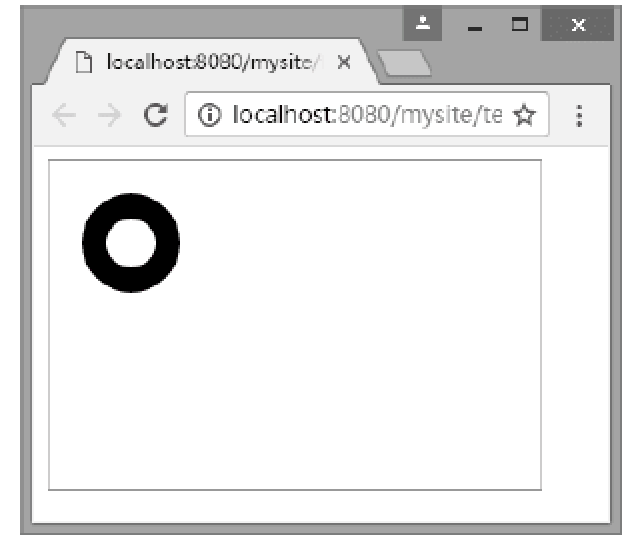


图 4.28 evenodd 规则填充

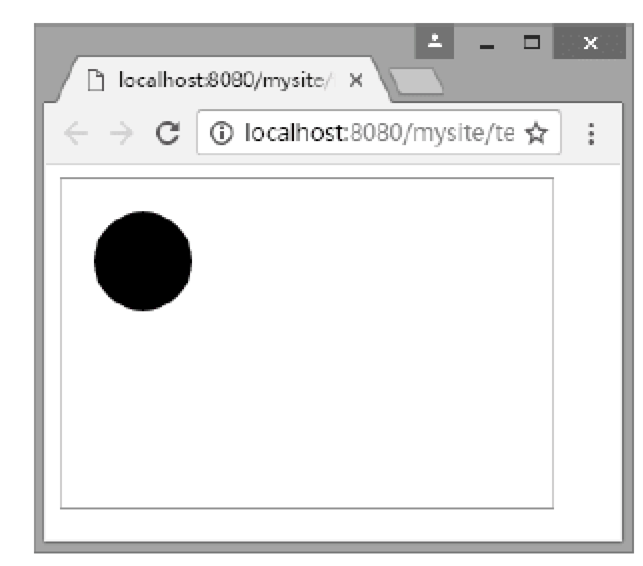


图 4.29 nonzero 规则填充

注意：IE 暂不支持 evenodd 规则填充。

4.4 图 形 变 形

本节将介绍如何对画布进行操作以及如何对画布中的图形进行变形，以便设计复杂图形。



视 频 讲 解

4.4.1 保存和恢复状态

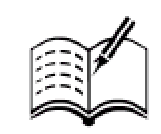
Canvas 状态存储在栈中，一个绘画状态包括两部分。

☑ 当前应用的变形，如移动、旋转和缩放，包括的样式属性：strokeStyle、fillStyle、globalAlpha、lineWidth、lineCap、lineJoin、miterLimit、shadowOffsetX、 shadowOffsetY、shadowBlur、shadowColor、globalCompositeOperation。

☑ 当前的裁切路径，参考 4.5 节介绍。

使用 save()方法，可以将当前的状态推送到栈中保存，使用 restore()方法可以将上一个保存的状态就从栈中弹出，恢复上一次所有的设置。





Note

【示例】下面示例先绘制一个矩形，填充颜色为#ff00ff，轮廓颜色为蓝色，然后保存这个状态，再绘制另外一个矩形，填充颜色为#ff0000，轮廓颜色为绿色，最后恢复第一个矩形的状态，并绘制两个小的矩形，则其中一个矩形填充颜色必为#ff00ff，另外矩形轮廓颜色必为蓝色，因为此时已经恢复了原来保存的状态，所以会沿用最先设定的属性值，预览效果如图 4.30 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    //开始绘制矩形
    ctx.fillStyle="#ff00ff";
    ctx.strokeStyle="blue";
    ctx.fillRect(20,20,100,100);
    ctx.strokeRect(20,20,100,100);
    ctx.fill();
    ctx.stroke();
    //保存当前 canvas 状态
    ctx.save();
    //绘制另外一个矩形
    ctx.fillStyle="#ff0000";
    ctx.strokeStyle="green";
    ctx.fillRect(140,20,100,100);
    ctx.strokeRect(140,20,100,100);
    ctx.fill();
    ctx.stroke();
    //恢复第一个矩形的状态
    ctx.restore();
    //绘制两个矩形
    ctx.fillRect(20,140,50,50);
    ctx.strokeRect(80,140,50,50);
}
```

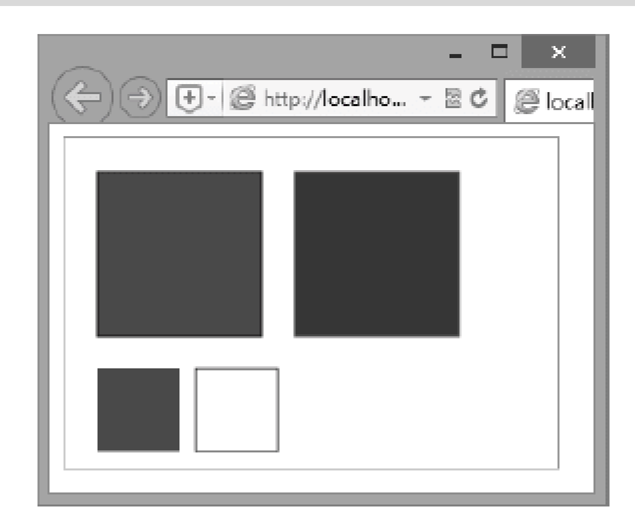


图 4.30　保存与恢复 canvas 状态

4.4.2　清除画布



视频讲解

使用 clearRect()方法可以清除指定区域内的所有图形，显示画布背景，该方法用法如下。

```
context.clearRect(x,y,width,height);
```

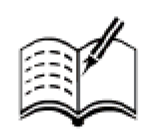
参数说明如下：

☑　x：要清除的矩形左上角的 x 坐标。

☑　y：要清除的矩形左上角的 y 坐标。

☑　width：要清除的矩形的宽度，以像素计。





☑　height：要清除的矩形的高度，以像素计。

【示例】下面示例演示了如何使用 clearRect()方法来擦除画布中的绘图。

```
<canvas id="canvas" width="300" height="200" style="border:solid 1px #999;"></canvas>
<input name="" type="button"   value="清空画布" onClick="clearMap();">

<script>
var ctx = document.getElementById('canvas').getContext('2d');
ctx.strokeStyle="#FF00FF";
ctx.beginPath();
ctx.arc(200,150,100,-Math.PI*1/6,-Math.PI*5/6,true);
ctx.stroke();
function clearMap(){
      ctx.clearRect(0,0,300,200);
}
</script>
```

在浏览器中的预览效果如图 4.31 所示，先是在画布上绘制一段弧线。如果单击“清空画布”按钮，则会清除这段弧线，如图 4.32 所示。



图 4.31　绘制弧线

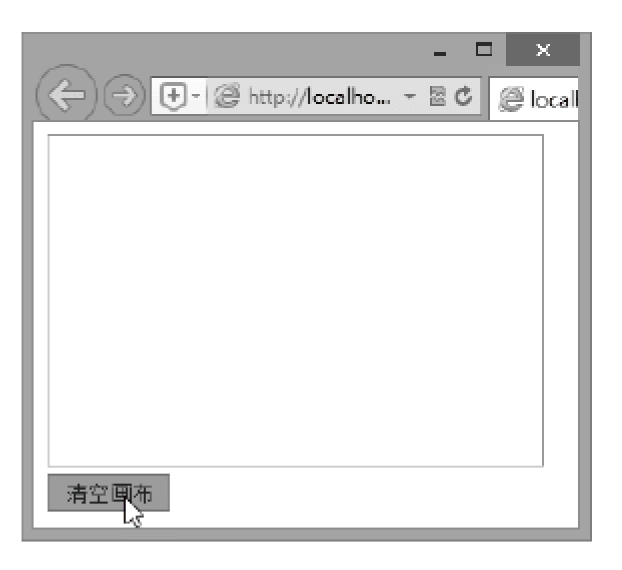


图 4.32　清空画布

4.4.3　移动坐标



在默认状态下，画布以左上角（0,0）为原点作为绘图参考。使用 translate()方法可以移动坐标原点，这样新绘制的图形就以新的坐标原点为参考进行绘制。其用法如下。

```
context.translate(dx, dy);
```

参数 dx 和 dy 分别为坐标原点沿水平和垂直两个方向的偏移量，如图 4.33 所示。

注意：在使用 translate()方法之前，应该先使用 save()方法保存画布的原始状态。当需要时可以使用 restore()方法恢复原始状态，特别是在重复绘图时非常重要。

【示例】下面示例综合运用了 save()、restore()、translate()方法来绘制一个伞状图形。

```
<canvas id="canvas" width="600" height="200" style="border:solid 1px #999;"></canvas>
<script>
draw();
function draw() {
      var ctx = document.getElementById('canvas').getContext('2d');
      //注意：所有的移动都是基于这一上下文
      ctx.translate(0,80);
```



Note

```
        for (var i=1;i<10;i++){
            ctx.save();
            ctx.translate(60*i, 0);
            drawTop(ctx,"rgb("+(30*i)+","+(255-30*i)+",255)");
            drawGrip(ctx);
            ctx.restore();
        }
    }
    //绘制伞形顶部半圆
    function drawTop(ctx, fillStyle){
        ctx.fillStyle = fillStyle;
        ctx.beginPath();
        ctx.arc(0, 0, 30, 0,Math.PI,true);
        ctx.closePath();
        ctx.fill();
    }
    //绘制伞形底部手柄
    function drawGrip(ctx){
        ctx.save();
        ctx.fillStyle = "blue";
        ctx.fillRect(-1.5, 0, 1.5, 40);
        ctx.beginPath();
        ctx.strokeStyle="blue";
        ctx.arc(-5, 40, 4, Math.PI,Math.PI*2,true);
        ctx.stroke();
        ctx.closePath();
        ctx.restore();
    }
    </script>
```

在浏览器中的预览效果如图 4.34 所示。可见，canvas 中图形移动的实现，其实是通过改变画布的坐标原点来实现的，所谓的“移动图形”，只是“看上去”的样子，实际移动的是坐标空间。领会并掌握这种方法，对于随心所欲地绘制图形非常有帮助。

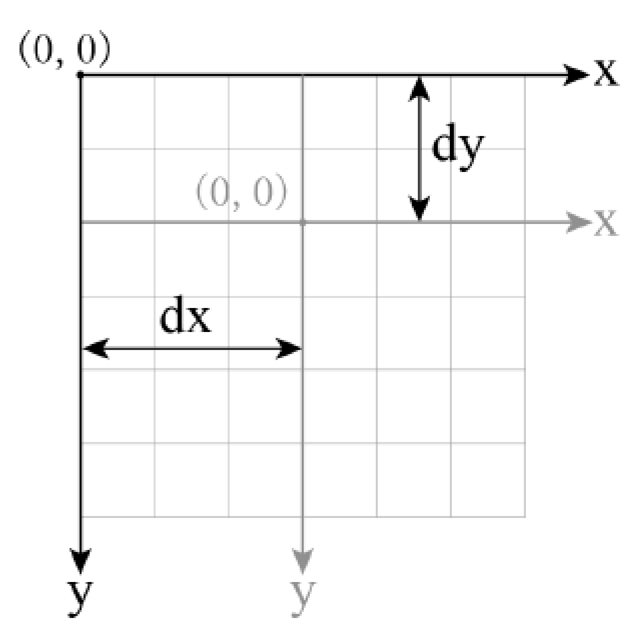




图 4.33　坐标空间的偏移示意图

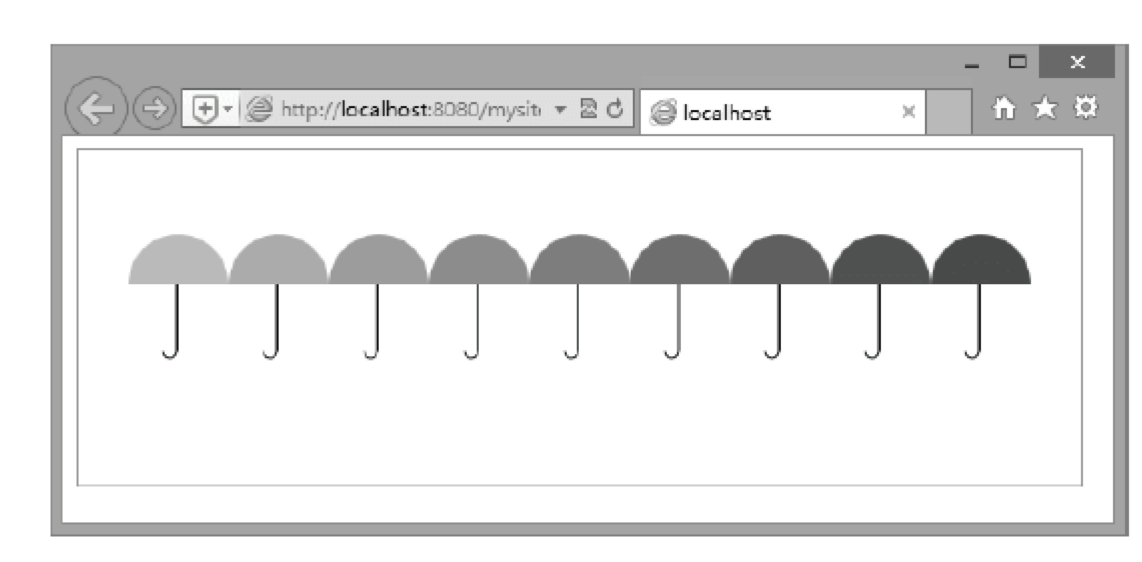


图 4.34　移动坐标空间

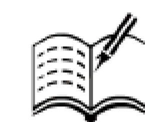


视频讲解

4.4.4　旋转坐标

使用 rotate()方法可以以原点为中心旋转 canvas 上下文对象的坐标空间，其用法如下。





```
context.rotate(angle);
```

rotate()方法只有一个参数，即旋转角度 angle，旋转角度以顺时针方向为正方向，以弧度为单位，旋转中心为 canvas 的原点，如图 4.35 所示。

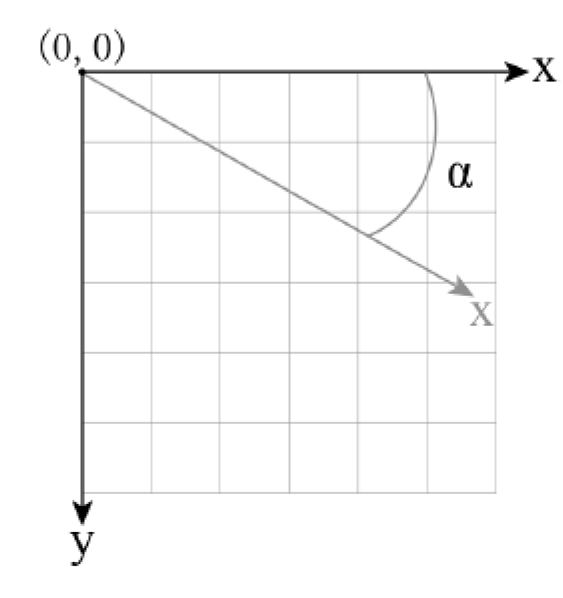


图 4.35　以原点为中心旋转 canvas

> 提示：如需将角度转换为弧度，可以使用 degrees*Math.PI/180 公式进行计算。例如，如果要旋转 5°，可套用这样的公式：5*Math.PI/180。

【示例】在上节示例的基础上，下面示例设计在每次开始绘制图形之前，先将坐标空间旋转 PI*(2/4+i/4)，再将坐标空间沿 y 轴负方向移动 100，然后开始绘制图形，从而实现使图形沿一中心点平均旋转分布。在浏览器中的预览效果如图 4.36 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.translate(150,150);
    for (var i=1;i<9;i++){
        ctx.save();
        ctx.rotate(Math.PI*(2/4+i/4));
        ctx.translate(0,-100);
        drawTop(ctx,"rgb("+(30*i)+","+(255-30*i)+",255)");
        drawGrip(ctx);
        ctx.restore();
    }
}
```

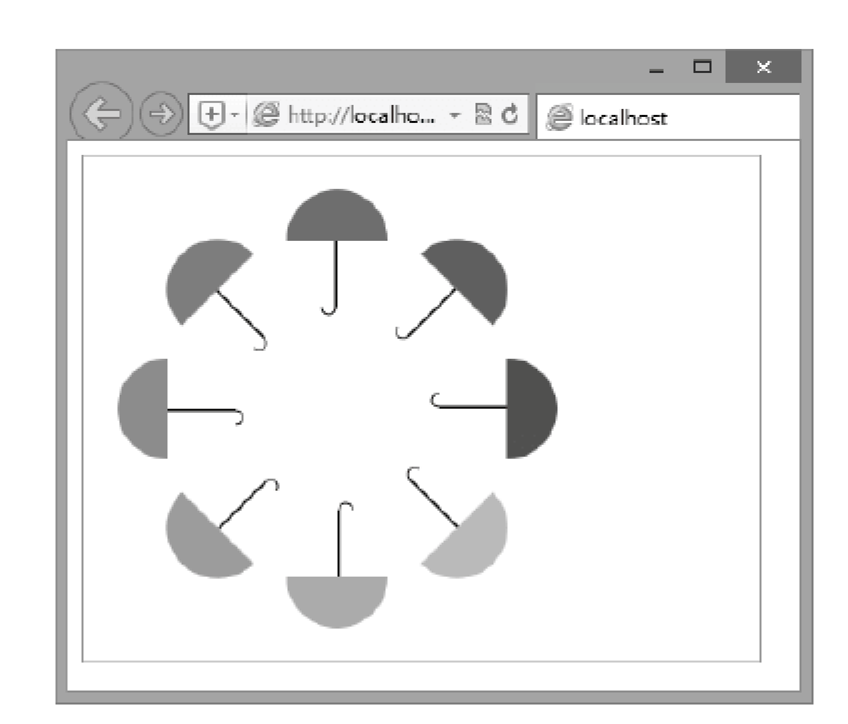
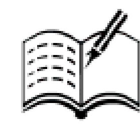


图 4.36　旋转坐标空间



视 频 讲 解



Note

4.4.5 缩放图形

使用 scale()方法可以增减 canvas 上下文对象的像素数目，从而实现图形的放大或缩小，其用法如下。

```
context.scale(x,y);
```

其中 x 为横轴的缩放因子，y 轴为纵轴的缩放因子，值必须是正值。如果需要放大图形，则将参数值设置为大于 1 的数值，如果需要缩小图形，则将参数值设置为小于 1 的数值，当参数值等于 1 时则没有任何效果。

【示例】下面示例使用 scale(0.95,0.95)来缩小图形到上次的 0.95，共循环 80 次，同时移动和旋转坐标空间，从而实现图形呈螺旋状由大到小的变化，预览效果如图 4.37 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.translate(200,20);
    for (var i=1;i<80;i++){
        ctx.save();
        ctx.translate(30,30);
        ctx.scale(0.95,0.95);
        ctx.rotate(Math.PI/12);
        ctx.beginPath();
        ctx.fillStyle="red";
        ctx.globalAlpha="0.4";
        ctx.arc(0,0,50,0,Math.PI*2,true);
        ctx.closePath();
        ctx.fill();
    }
}
```

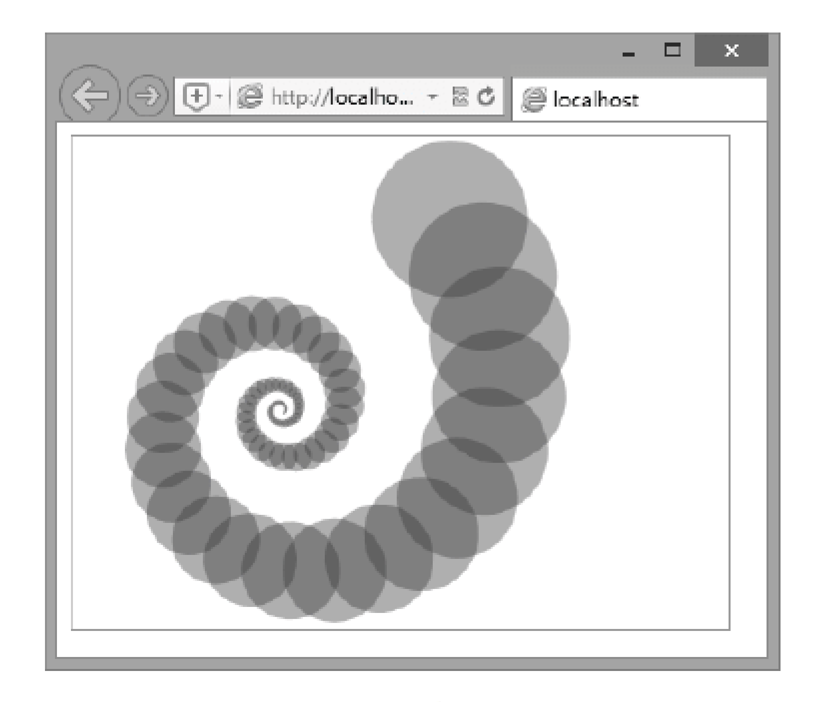


图 4.37　缩放图形



视 频 讲 解

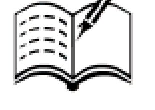
4.4.6 变换图形

transform()方法可以同时缩放、旋转、移动和倾斜当前的上下文环境，用法如下所示：

```
context.transform(a,b,c,d,e,f);
```

参数说明如下：





☑　a：水平缩放绘图。
☑　b：水平倾斜绘图。
☑　c：垂直倾斜绘图。
☑　d：垂直缩放绘图。
☑　e：水平移动绘图。
☑　f：垂直移动绘图。

提示：☑　translate(x,y)可以用下面的方法来代替：

context.transform(0,1,1,0,dx,dy);

或：

context.transform(1,0,0,1,dx,dy);

其中 dx 为原点沿 x 轴移动的数值，dy 为原点沿 y 轴移动的数值。

☑　scale(x,y)可以用下面的方法来代替：

context.transform(m11,0,0,m22,0,0);

或：

context.transform(0,m12,m21,0,0,0);

其中 dx、dy 都为 0 表示坐标原点不变。m11、m22 或 m12、m21 为沿 x、y 轴放大的倍数。

☑　rotate(angle)可以用下面的方法来代替：

context.transform(cosθ,sinθ,- sinθ, cosθ,0,0);

其中的 θ 为旋转角度的弧度值，dx、dy 都为 0 表示坐标原点不变。

setTransform()方法用于将当前的变换矩阵进行重置为最初的矩阵，然后以相同的参数调用 transform 方法，用法如下所示。

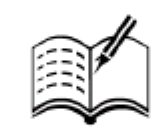
```
context.setTransform(m11, m12, m21, m22, dx, dy);
```

【示例】下面示例使用 setTransform()方法将前面已经发生变换的矩阵首先重置为最初的矩阵，即恢复最初的原点，然后再将坐标原点改为（10,10），并以新的坐标为基准绘制一个蓝色的矩形。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.translate(200,20);
    for (var i=1;i<90;i++){
        ctx.save();
        ctx.transform(0.95,0,0,0.95,30,30);
        ctx.rotate(Math.PI/12);
        ctx.beginPath();
        ctx.fillStyle="red";
        ctx.globalAlpha="0.4";
        ctx.arc(0,0,50,0,Math.PI*2,true);
        ctx.closePath();
        ctx.fill();
    }
    ctx.setTransform(1,0,0,1,10,10);
```



```
        ctx.fillStyle="blue";
        ctx.fillRect(0,0,50,50);
        ctx.fill();
    }
```


Note

在浏览器中的预览效果如图 4.38 所示。在本例中，使用 scale(0.95,0.95)来缩小图形到上次的 0.95，共循环 89 次，同时移动和旋转坐标空间，从而实现图形呈螺旋状由大到小的变化。



图 4.38　矩阵重置并变换

4.5　图形合成


视频讲解

本节将介绍图形合成的一般方法，以及路径裁切的实现。

4.5.1　合成

当两个或两个以上的图形存在重叠区域时，默认情况下一个图形画在前一个图形之上。通过指定图形 globalCompositeOperation 属性的值可以改变图形的绘制顺序或绘制方式，从而实现更多种可能。

【示例】下面示例设置所有图形的透明度为 1，即不透明。设置 globalCompositeOperation 属性值为 source-over，即默认设置，新的图形会覆盖在原有图形之上，也可以指定其他值，详见表 4.1。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.fillStyle="red";
    ctx.fillRect(50,25,100,100);
    ctx.fillStyle="green";
    ctx.globalCompositeOperation="source-over";
    ctx.beginPath();
    ctx.arc(150,125,50,0,Math.PI*2,true);
    ctx.closePath();
    ctx.fill();
}
```

在浏览器中的预览效果如图 4.39 所示。如果将 globalAlpha 的值更改为 0.5（ctx.globalAlpha=0.5;），则两个图形都会呈半透明效果，如图 4.40 所示。



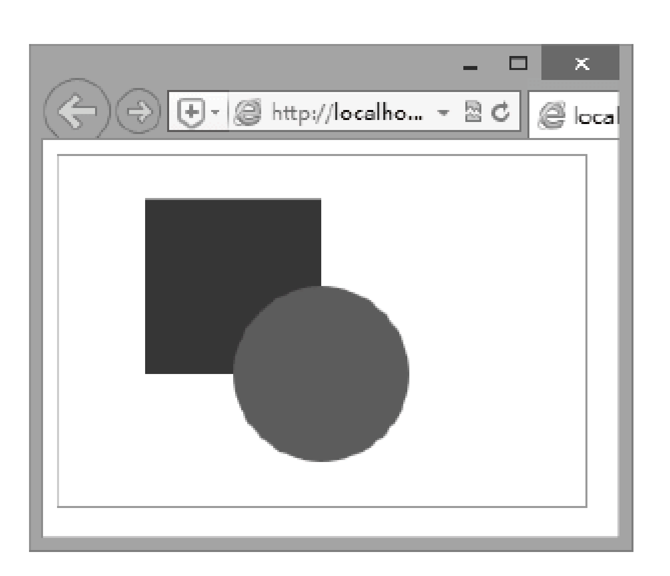



图 4.39　图形的组合

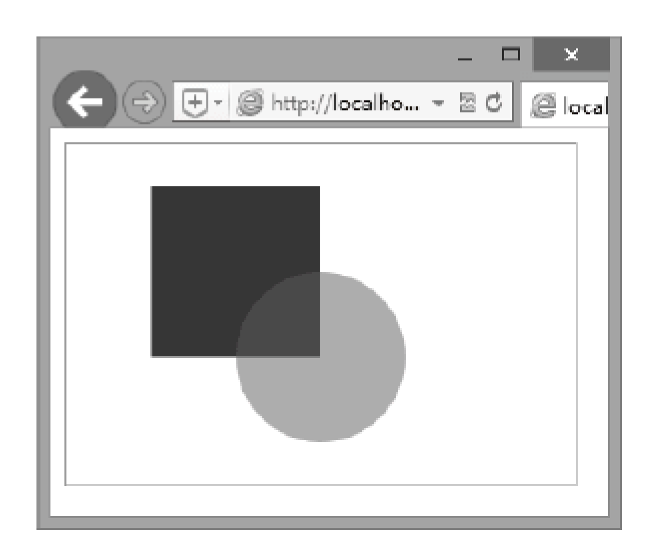


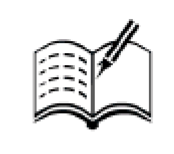
图 4.40　半透明效果

表 4.1 给出了 globalCompositeOperation 属性所有可用的值。表中的图例矩形表示为 B，为先绘制的图形（原有内容为 destination），圆形表示为 A，为后绘制的图形（新图形为 source）。在应用时注意 globalCompositeOperation 语句的位置，应处在原有内容与新图形之间。Chrome 浏览器支持大多数属性值，无效的在表中已经标出。Opera 浏览器对这些属性值的支持相对来说更好一些。

表 4.1　globalCompositeOperation 属性所有可用的值

属　性　值	图形合成示例	说　　明
source-over（默认值）		A over B，这是默认设置，即新图形覆盖在原有内容之上
destination-over		B over A，即原有内容覆盖在新图形之上
source-atop		只绘制原有内容和新图形与原有内容重叠的部分，且新图形位于原有内容之上
destination-atop		只绘制新图形和新图形与原有内容重叠的部分，且原有内容位于重叠部分之下
source-in		新图形只出现在与原有内容重叠的部分，其余区域变为透明
destination-in		原有内容只出现在与新图形重叠的部分，其余区域为透明
source-out		新图形中与原有内容不重叠的部分被保留







续表

属　性　值	图形合成示例	说　　明
destination-out		原有内容中与新图形不重叠的部分被保留
lighter		两图形重叠的部分做加色处理
darker		两图形重叠的部分做减色处理
copy		只保留新图形。在Chrome浏览器中无效，Opera 11.5中有效
xor		将重叠的部分变为透明

4.5.2　裁切



视频讲解

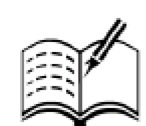
使用 clip()方法能够从原始画布中剪切任意形状和尺寸。其原理与绘制普通 canvas 图形类似，只不过 clip()的作用是形成一个蒙版，没有被蒙版的区域会被隐藏。

提示： 一旦剪切了某个区域，则所有之后的绘图都会被限制在被剪切的区域内，不能访问画布上的其他区域。用户也可以在使用 clip()方法前，通过使用 save()方法对当前画布区域进行保存，并在以后的任意时间通过 restore()方法对其进行恢复。

【示例】如果绘制一个圆形，并进行裁切，则圆形之外的区域将不会绘制在 canvas 上。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    //绘制背景
    ctx.fillStyle="black";
    ctx.fillRect(0,0,300,300);
    ctx.fill();
    //绘制圆形
    ctx.beginPath();
    ctx.arc(150,150,100,0,Math.PI*2,true);
    //裁切路径
    ctx.clip();
    ctx.translate(200,20);
    for (var i=1;i<90;i++){
        ctx.save();
```





```
            ctx.transform(0.95,0,0,0.95,30,30);
            ctx.rotate(Math.PI/12);
            ctx.beginPath();
            ctx.fillStyle="red";
            ctx.globalAlpha="0.4";
            ctx.arc(0,0,50,0,Math.PI*2,true);
            ctx.closePath();
            ctx.fill();
        }
}
```

可以看到只有圆形区域内螺旋图形被显示了出来，其余部分被裁切掉了，效果如图 4.41 所示。



图 4.41　裁切图形

4.6 绘制文本

使用 fillText()和 strokeText()方法，可以分别以填充方式和轮廓方式绘制文本。

4.6.1 填充文字



视频讲解

fillText()方法能够在画布上绘制填色文本，默认颜色是黑色。其用法如下。

```
context.fillText(text,x,y,maxWidth);
```

参数说明如下：

- ☑ text：规定在画布上输出的文本。
- ☑ x：开始绘制文本的 x 坐标位置（相对于画布）。
- ☑ y：开始绘制文本的 y 坐标位置（相对于画布）。
- ☑ maxWidth：可选。允许的最大文本宽度，以像素计。

【示例】下面使用 fillText()方法在画布上绘制文本“Hi”和“Canvas API”，效果如图 4.42 所示。

```
function draw() {
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    ctx.font="40px Georgia";
    ctx.fillText("Hi",10,50);
```



```
        ctx.font="50px Verdana";
        //创建渐变
        var gradient=ctx.createLinearGradient(0,0,canvas.width,0);
        gradient.addColorStop("0","magenta");
        gradient.addColorStop("0.5","blue");
        gradient.addColorStop("1.0","red");
        //用渐变填色
        ctx.fillStyle=gradient;
        ctx.fillText("Canvas API",10,120);
}
```



Note

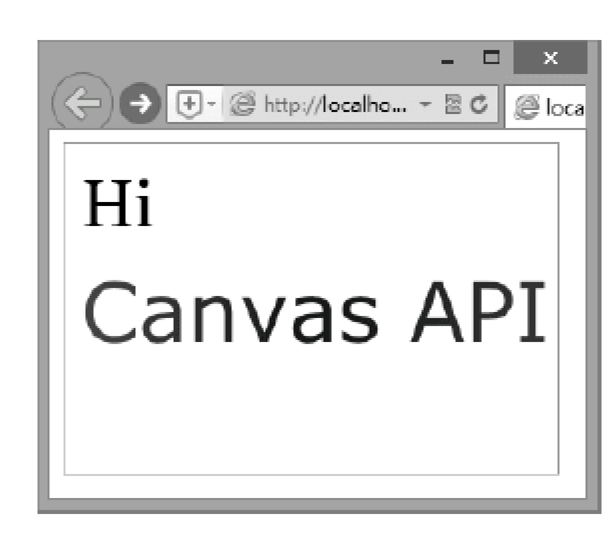


图 4.42　绘制填充文字

视频讲解

4.6.2　轮廓文字

使用 strokeText()方法可以在画布上绘制描边文本，默认颜色是黑色。其用法如下。

```
context.strokeText(text,x,y,maxWidth);
```

参数说明如下：

☑　text：规定在画布上输出的文本。

☑　x：开始绘制文本的 x 坐标位置（相对于画布）。

☑　y：开始绘制文本的 y 坐标位置（相对于画布）。

☑　maxWidth：可选。允许的最大文本宽度，以像素计。

【示例】下面使用 strokeText()方法绘制文本“Hi”和“Canvas API”，效果如图 4.43 所示。

```
function draw() {
        var canvas = document.getElementById('canvas');
        var ctx = canvas.getContext('2d');
        ctx.font="40px Georgia";
        ctx.fillText("Hi",10,50);
        ctx.font="50px Verdana";
        //创建渐变
        var gradient=ctx.createLinearGradient(0,0,canvas.width,0);
        gradient.addColorStop("0","magenta");
        gradient.addColorStop("0.5","blue");
        gradient.addColorStop("1.0","red");
        //用渐变填色
        ctx.strokeStyle=gradient;
        ctx.strokeText("Canvas API",10,120);
}
```



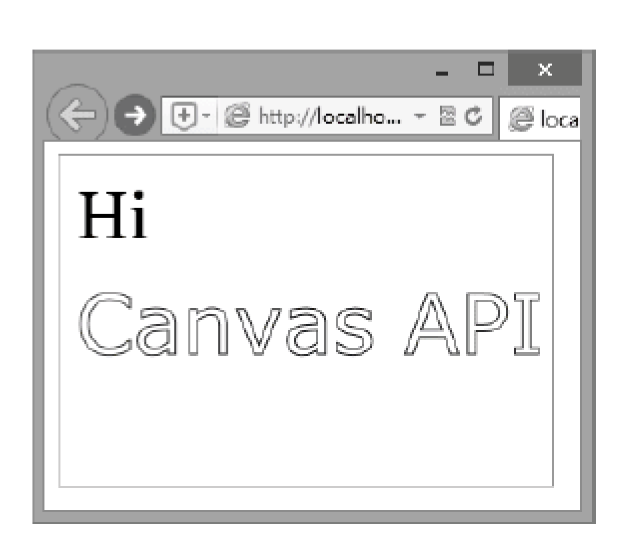
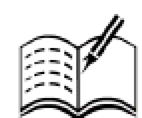


图 4.43　绘制轮廓文字





4.6.3　文本样式

下面简单介绍文本样式的相关属性。

☑　font：定义字体样式，语法与 CSS 字体样式相同。默认字体样式为 10px sans-serif。

☑　textAlign：设置正在绘制的文本水平对齐方式。取值说明如下：

- start：默认，文本在指定的位置开始。
- end：文本在指定的位置结束。
- center：文本的中心被放置在指定的位置。
- left：文本左对齐。
- right：文本右对齐。

☑　textBaseline：设置正在绘制的文本基线对齐方式，即文本垂直对齐方式。取值说明如下：

- alphabetic：默认值，文本基线是普通的字母基线。
- top：文本基线是 em 方框的顶端。
- hanging：文本基线是悬挂基线。
- middle：文本基线是 em 方框的正中。
- ideographic：文本基线是表意基线。
- bottom：文本基线是 em 方框的底端。

提示：大部分浏览器尚不支持 hanging 和 ideographic 属性值。

☑　direction：设置文本方向。取值说明如下：

- ltr：从左到右。
- rtl：从右到左。
- inherit，默认值，继承文本方向。

【示例 1】下面示例在 x 轴 150px 的位置创建一条竖线。位置 150 就被定义为所有文本的锚点。然后比较每种 textAlign 属性值对齐效果，如图 4.44 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    //在位置 150 创建一条竖线
    ctx.strokeStyle="blue";
    ctx.moveTo(150,20);
    ctx.lineTo(150,170);
    ctx.stroke();
    ctx.font="15px Arial";
    //显示不同的 textAlign 值
```



Note

```
        ctx.textAlign="start";
        ctx.fillText("textAlign=start",150,60);
        ctx.textAlign="end";
        ctx.fillText("textAlign=end",150,80);
        ctx.textAlign="left";
        ctx.fillText("textAlign=left",150,100);
        ctx.textAlign="center";
        ctx.fillText("textAlign=center",150,120);
        ctx.textAlign="right";
        ctx.fillText("textAlign=right",150,140);
    }
```

【示例 2】下面示例在 y 轴 100px 的位置创建一条水平线。位置 100 就被定义为用蓝色填充的矩形。然后比较每种 textBaseline 属性值对齐效果，如图 4.45 所示。

```
    function draw() {
        var ctx = document.getElementById('canvas').getContext('2d');
        //在位置 y=100 绘制蓝色线条
        ctx.strokeStyle="blue";
        ctx.moveTo(5,100);
        ctx.lineTo(395,100);
        ctx.stroke();
        ctx.font="20px Arial"
        //在 y=100 以不同的 textBaseline 值放置每个单词
        ctx.textBaseline="top";
        ctx.fillText("Top",5,100);
        ctx.textBaseline="bottom";
        ctx.fillText("Bottom",50,100);
        ctx.textBaseline="middle";
        ctx.fillText("Middle",120,100);
        ctx.textBaseline="alphabetic";
        ctx.fillText("Alphabetic",190,100);
        ctx.textBaseline="hanging";
        ctx.fillText("Hanging",290,100);
    }
```

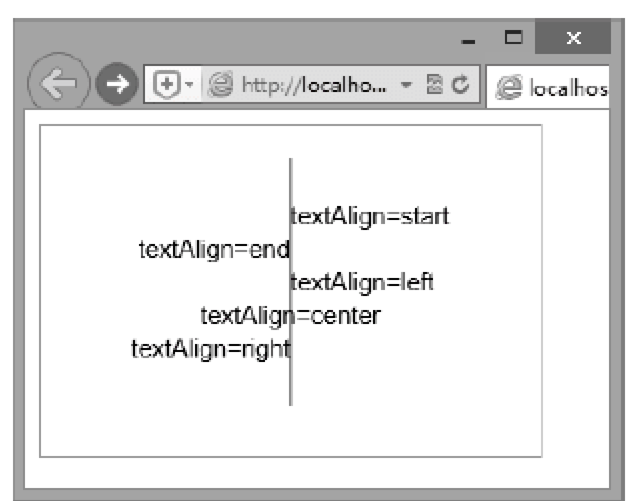


图 4.44　比较每种 textAlign 属性值对齐效果

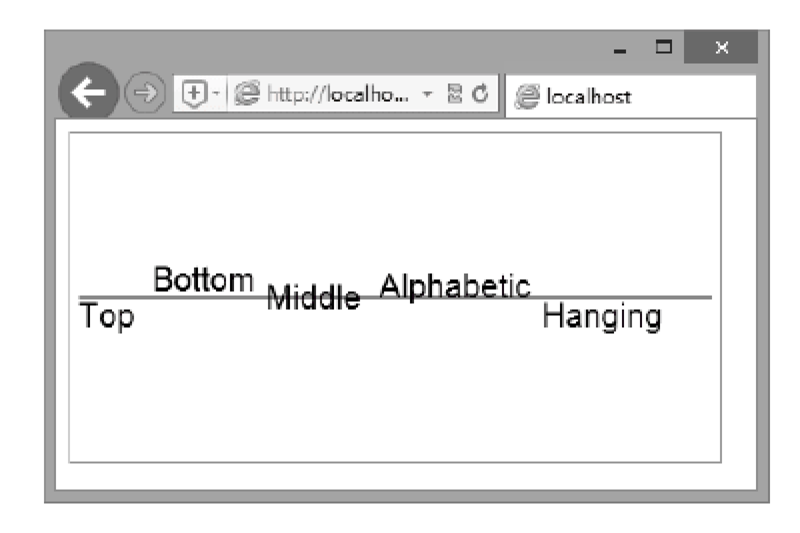


图 4.45　比较每种 textBaseline 属性值对齐效果



视 频 讲 解

4.6.4　测量宽度

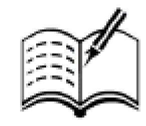
使用 measureText()方法可以测量当前所绘制文字中指定文字的宽度，它返回一个 TextMetrics 对象，使用该对象的 width 属性可以得到指定文字参数后所绘制文字的总宽度，其用法如下。



```
metrics=context. measureText(text);
```

其中的参数 text 为要绘制的文字。


提示：如果需要在文本向画布输出之前就了解文本的宽度，应该使用该方法。



【示例】下面是测量文字宽度的一个示例，预览效果如图 4.46 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    ctx.font = "bold 20px  楷体";
    ctx.fillStyle="Blue";
    var txt1 = "HTML5+CSS3";
    ctx.fillText(txt1,10,40);
    var txt2 = "以上字符串的宽度为：";
    var mtxt1 = ctx.measureText(txt1);
    var mtxt2 = ctx.measureText(txt2);
    ctx.font = "bold 15px  宋体";
    ctx.fillStyle="Red";
    ctx.fillText(txt2,10,80);
    ctx.fillText(mtxt1.width,mtxt2.width,80);
}
```



图 4.46　测量文字宽度

4.7　使用图像

在 canvas 中可以导入图像。导入的图像可以改变大小、裁切或合成。canvas 支持多种图像格式，如 PNG、GIF、JPEG 等。

4.7.1　导入图像



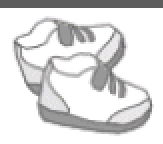
在 canvas 中导入图像的步骤：

第 1 步，确定图像来源。

第 2 步，使用 drawImage()方法将图像绘制到 canvas 中。

确定图像来源有 4 种方式，用户任选一种即可。

☑ 页面内的图片：如果已知图片元素的 ID，则可以通过 document.images 集合、document.getElementsByTagName() 或 document.getElementById()等方法获取页面内的该图片元素。





Note

☑ 其他 canvas 元素：可以通过 document.getElementsByTagName()或 document.getElementById()等方法获取已经设计好的 canvas 元素。例如，可以用这种方法为一个比较大的 canvas 生成缩略图。

☑ 用脚本创建一个新的 image 对象：使用脚本可以从零开始创建一个新的 image 对象。
不过这种方法存在一个缺点：如果图像文件来源于网络且较大，则会花费较长的时间来装载。所以如果不希望因为图像文件装载而导致漫长的等待，需要做好预装载的工作。

☑ 使用 data:url 方式引用图像：这种方法允许用 Base64 编码的字符串来定义一个图片，优点是图片可以即时使用，不必等待装载，而且迁移也非常容易。缺点是无法缓存图像，所以如果图片较大，则不太适宜用这种方法，因为这会导致嵌入的 url 数据相当庞大。

使用脚本创建新 image 对象时，其方法如下所示。

```
var img = new Image();    //创建新的 Image 对象
img.src = 'image1.png';   //设置图像路径
```

如果要解决图片预装载的问题，则可以使用下面的方法，即使用 onload 事件一边装载图像一边执行绘制图像的函数。

```
var img = new Image();    //创建新的 Image 对象
img.onload = function(){
    //此处放置 drawImage 的语句
}
img.src = 'image1.png';   //设置图像路径
```

不管采用什么方式获取图像来源，之后的工作都是使用 drawImage()方法将图像绘制到 canvas 中。drawImage()方法能够在画布上绘制图像、画布或视频。该方法也能够绘制图像的某些部分，以及增加或减少图像的尺寸。其用法如下所示。

```
//语法 1：在画布上定位图像
context.drawImage(img,x,y);
//语法 2：在画布上定位图像，并规定图像的宽度和高度
context.drawImage(img,x,y,width,height);
//语法 3：剪切图像，并在画布上定位被剪切的部分
context.drawImage(img,sx,sy,swidth,sheight,x,y,width,height);
```

参数说明如下：

☑ img：规定要使用的图像、画布或视频。
☑ sx：可选。开始剪切的 x 坐标位置。
☑ sy：可选。开始剪切的 y 坐标位置。
☑ swidth：可选。被剪切图像的宽度。
☑ sheight：可选。被剪切图像的高度。
☑ x：在画布上放置图像的 x 坐标位置。
☑ y：在画布上放置图像的 y 坐标位置。
☑ width：可选。要使用的图像的宽度。可以实现伸展或缩小图像。
☑ height：可选。要使用的图像的高度。可以实现伸展或缩小图像。

【示例】下面示例演示了如何使用上述步骤将图像引入 canvas 中，预览效果如图 4.47 所示。至于第二和第三种 drawImage()方法，我们将在后续小节中单独介绍。

```
function draw() {
```



```
        var ctx = document.getElementById('canvas').getContext('2d');
        var img = new Image();
        img.onload = function(){
            ctx.drawImage(img,0,0);
        }
        img.src = 'images/1.jpg';
    }
```



Note



图 4.47 向 canvas 中导入图像

4.7.2 缩放图像

drawImage()方法的第二种用法可以用于使图片按指定的大小显示，其用法如下。

```
context.drawImage(image, x, y, width, height);
```

其中 width 和 height 分别是图像在 canvas 中显示的宽度和高度。

【示例】 下面示例将 4.7.1 节示例中的代码稍作修改，设置导入的图像放大显示，并仅显示头部位置，效果如图 4.48 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var img = new Image();
    img.onload = function(){
        ctx.drawImage(img,-100,-40,800,500);
    }
    img.src = 'images/1.jpg';
}
```

视 频 讲 解

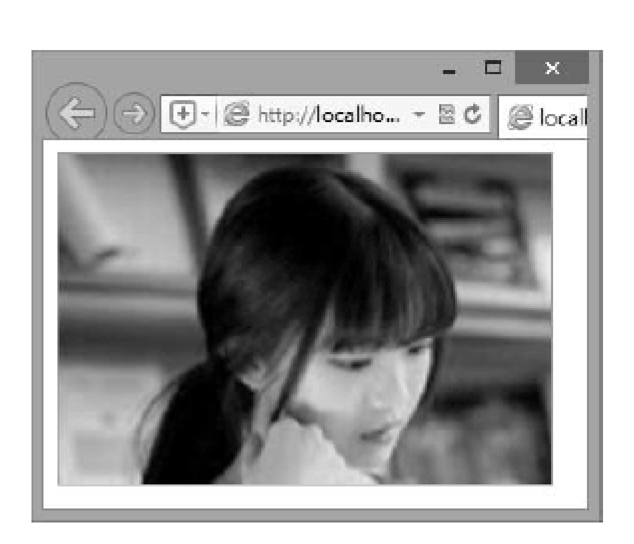


图 4.48 放大图像显示

4.7.3 裁切图像

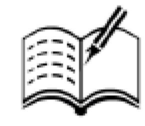
drawImage 的第三种用法用于创建图像切片，其用法如下。



视 频 讲 解

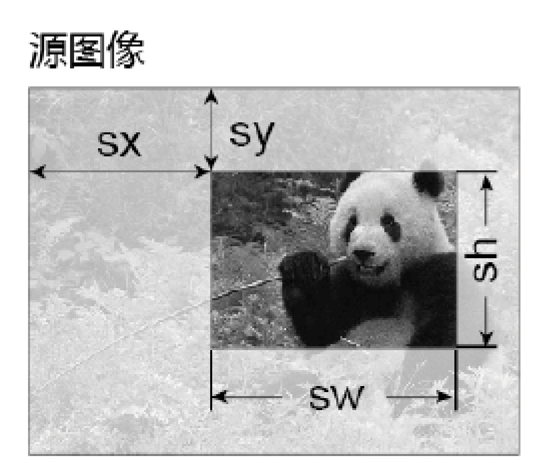


```
context.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh);
```



Note

其中 image 参数与前两种用法相同，其余 8 个参数可以参考下面的图示。sx、sy 为源图像被切割区域的起始坐标，sw、sh 为源图像被切下来的宽度和高度，dx、dy 为被切割下来的源图像要放置到目标 canvas 的起始坐标，dw、dh 为被切割下来的源图像放置到目标 canvas 的显示宽度和高度，如图 4.49 所示。



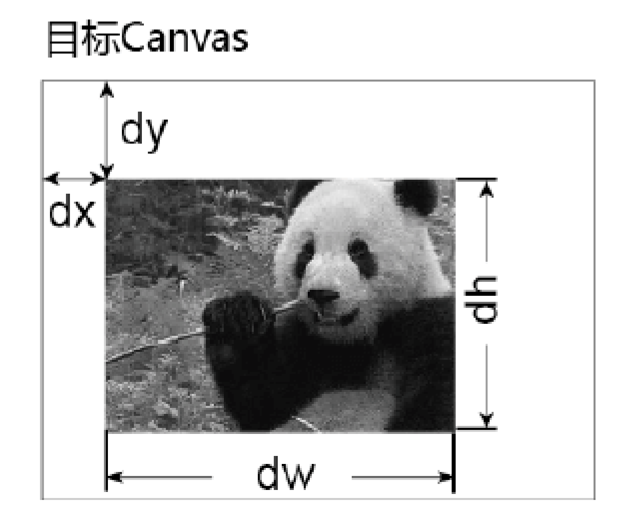


图 4.49　其余 8 个参数的图示

【示例】下面示例演示如何创建图像切片，预览效果如图 4.50 所示。

```
function draw() {
    var ctx = document.getElementById('canvas').getContext('2d');
    var img = new Image();
    img.onload = function(){
        ctx.drawImage(img,70,50,100,70,5,5,290,190);
    }
    img.src = 'images/1.jpg';
}
```

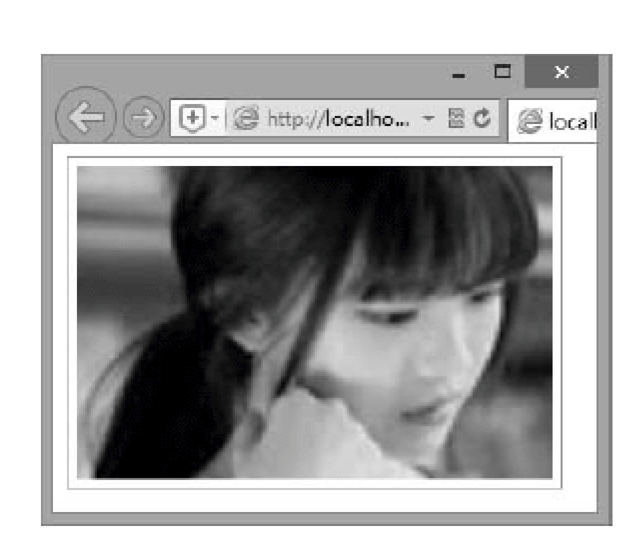


图 4.50　创建图像切片



视 频 讲 解

4.7.4　平铺图像

图像平铺就是让图像填满画布，有两种方法可以实现，下面结合示例进行说明。

【示例 1】第一种方法是使用 drawImage()方法。

```
function draw() {
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    var image = new Image();
    image.src = "images/1.png";
    image.onload = function(){
        var scale=5                          //平铺比例
```



Note

```
            var n1=image.width/scale;               //缩小后图像宽度
            var n2=image.height/scale;              //缩小后图像高度
            var n3=canvas.width/n1;                 //平铺横向个数
            var n4=canvas.height/n2;                //平铺纵向个数
            for(var i=0;i<n3;i++)
                for(var j=0;j<n4;j++)
                    ctx.drawImage(image,i*n1,j*n2,n1,n2);
        };
}
```

本例用到几个变量以及循环语句，相对来说处理方法复杂一些，预览效果如图 4.51 所示。

【示例 2】使用 createPattern()方法，该方法只使用了几个参数就达到了上面所述的平铺效果。createPattern()方法的用法如下所示：

```
context.createPattern(image,type);
```

参数 image 为要平铺的图像，参数 type 必须是下面的字符串值之一：

☑　no-repeat：不平铺。

☑　repeat-x：横方向平铺。

☑　repeat-y：纵方向平铺。

☑　repeat：全方向平铺。

创建 image 对象，指定图像文件后，使用 createPattern()方法创建填充样式，然后将该样式指定给图形上下文对象的 fillStyle 属性，最后填充画布，重复填充的效果如图 4.52 所示。

```
function draw() {
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    var image = new Image();
    image.src = "images/1.png";
    image.onload = function(){
        var ptrn = ctx.createPattern(image,'repeat');       //创建填充样式，全方向平铺
        ctx.fillStyle = ptrn;                               //指定填充样式
        ctx.fillRect(0,0,300,200);                          //填充画布
    };
}
```

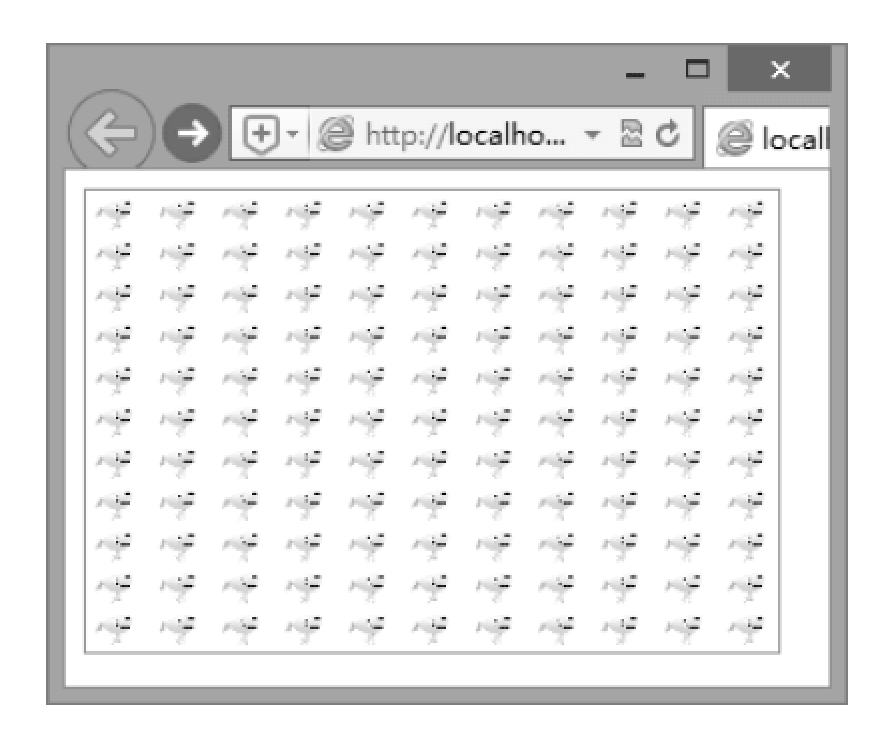


图 4.51　通过 drawImage()方法平铺显示

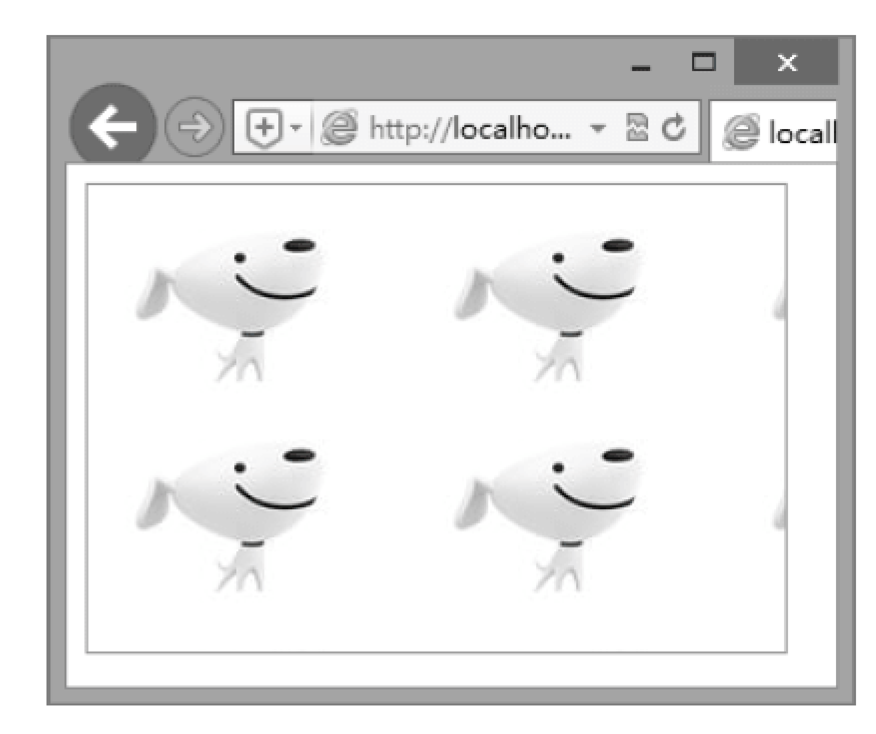
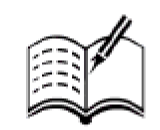


图 4.52　通过 createPattern()方法平铺显示




4.8 像素操作


Note

到目前为止，我们尚未深入了解 canvas 画布真实像素的原理，事实上，用户可以直接通过 ImageData 对象操纵像素数据，直接读取或将数据数组写入该对象中。

4.8.1 认识 ImageData 对象

ImageData 对象表示图像数据，存储 canvas 对象真实的像素数据，它包含以下几个只读属性：

☑ width：返回 ImageData 对象的宽度，单位是像素。

☑ height：返回 ImageData 对象的高度，单位是像素。

☑ data：返回一个对象，其包含指定的 ImageData 对象的图像数据。

图像数据是一个数组，包含着 RGBA 格式的整型数据，范围在 0 至 255 之间（包括 255），通过图像数据可以查看画布初始像素数据。每个像素用 4 个值来代表，分别是红、绿、蓝和透明值。对于透明值来说，0 是透明的，255 是完全可见的。数组格式如下：

```
[r1, g1, b1, a1, r2, g2, b2, a2, r3, g3, b3, a3,…]
```

r1、g1、b1 和 a1 分别为第一个像素的红色值、绿色值、蓝色值和透明度值。r2、g2、b2、a2 分别为第二个像素的红色值、绿色值、蓝色值、透明度值，依此类推。像素是从左到右，然后自上而下，使用 data.length 可以遍历整个数组。

4.8.2 创建图像数据

使用 createImageData()方法可以创建一个新的、空白的 ImageData 对象，具体用法如下所示。

```
//以指定的尺寸（以像素计）创建新的 ImageData 对象
var imgData=context.createImageData(width,height);
//创建与指定的另一个 ImageData 对象尺寸相同的新 ImageData 对象（不会复制图像数据）
var imgData=context.createImageData(imageData);
```

参数简单说明如下：

☑ width：定义 ImageData 对象的宽度，以像素计。

☑ height：定义 ImageData 对象的高度，以像素计。

☑ imageData：指定另一个 ImageData 对象。

调用该方法将创建一个指定大小的 ImageData 对象，所有像素被预设为透明黑。


视频讲解

4.8.3 将图像数据写入画布

putImageData()方法可以将图像数据从指定的 ImageData 对象写入画布。具体用法如下。

```
context.putImageData(imgData,x,y,dirtyX,dirtyY,dirtyWidth,dirtyHeight);
```

参数简单说明如下：

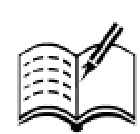
☑ imgData：要写入画布的 ImageData 对象。

☑ x：ImageData 对象左上角的 x 坐标，以像素计。

☑ y：ImageData 对象左上角的 y 坐标，以像素计。



☑ dirtyX：可选参数，在画布上放置图像的 x 轴位置，以像素计。
☑ dirtyY：可选参数，在画布上放置图像的 y 轴位置，以像素计。
☑ dirtyWidth：可选参数，在画布上绘制图像所使用的宽度。
☑ dirtyHeight：可选参数，在画布上绘制图像所使用的高度。



Note

【示例】下面示例创建一个 100×100 像素的 ImageData 对象，其中每个像素都是红色的，然后把它写入画布中显示出来。

```
<canvas id="myCanvas"></canvas>
<script>
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var imgData=ctx.createImageData(100,100);          //创建图像数据
//使用 for 循环语句，逐一设置图像数据中每个像素的颜色值
for (var i=0;i<imgData.data.length;i+=4){
      imgData.data[i+0]=255;
      imgData.data[i+1]=0;
      imgData.data[i+2]=0;
      imgData.data[i+3]=255;
}
ctx.putImageData(imgData,10,10);                   //把图像数据写入画布
</script>
```

4.8.4 在画布中复制图像数据



视频讲解

getImageData()方法能复制画布指定矩形的像素数据，返回 ImageData 对象，用法如下所示。

```
var imgData=context.getImageData(x,y,width,height);
```

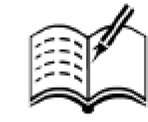
参数简单说明如下：

☑ x：开始复制的左上角位置的 x 坐标。
☑ y：开始复制的左上角位置的 y 坐标。
☑ width：将要复制的矩形区域的宽度。
☑ height：将要复制的矩形区域的高度。

【示例】下面示例先创建一个图像对象，使用 src 属性加载外部图像源，加载成功之后，使用 drawImage()方法把外部图像绘制到画布上。然后使用 getImageData()方法把画布中的图像转换为 ImageData（图像数据）对象。然后，使用 for 语句逐一访问每个像素点，对每个像素的颜色进行反显操作，再存回数组。最后，使用 putImageData()方法将反显操作后的图像重绘在画布上。

```
<canvas id="myCanvas" width="384" height="240"></canvas>
<script>
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext('2d');
var image = new Image();
image.src = "images/1.jpg";
image.onload = function (){
      context.drawImage(image, 0, 0);
      var imagedata = context.getImageData(0,0,image.width,image.height);
      for (var i = 0, n = imagedata.data.length; i < n; i += 4){
            imagedata.data[i+0] = 255 - imagedata.data[i+0]; // red
```





Note

```
            imagedata.data[i+1] = 255 - imagedata.data[i+2]; // green
            imagedata.data[i+2] = 255 - imagedata.data[i+1]; // blue
        }
        context.putImageData(imagedata, 0, 0);
    };
    </script>
```

以上代码在 IE 浏览器中的预览效果如图 4.53 所示。





（a）原图



（b）反转效果图

图 4.53　图像反色显示



视 频 讲 解

4.8.5　保存图片

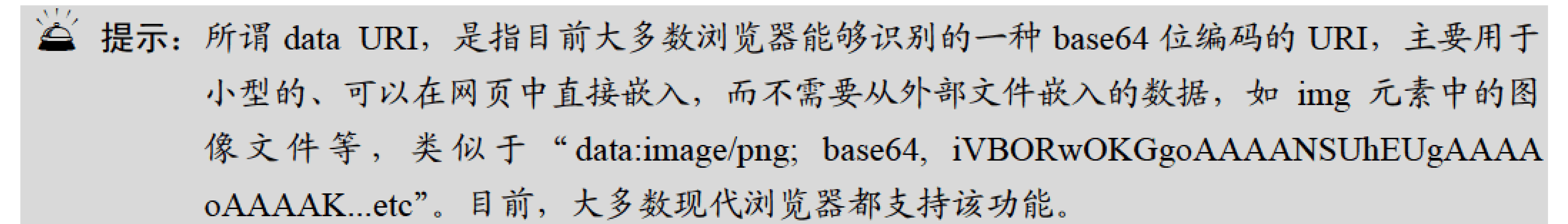
HTMLCanvasElement 提供一个 toDataURL()方法，使用它可以将画布保存为图片，返回一个包含图片展示的 data URI。具体用法如下：

```
canvas.toDataURL(type, encoderOptions);
```

参数简单说明如下：

☑　type：可选参数，默认为 image/png。

☑　encoderOptions：可选参数，默认为 0.92。在指定图片格式为 image/jpeg 或 image/webp 的情况下，可以设置图片的质量，取值从 0 到 1 的区间内选择，如果超出取值范围，将会使用默认值。



提示：所谓 data URI，是指目前大多数浏览器能够识别的一种 base64 位编码的 URI，主要用于小型的、可以在网页中直接嵌入，而不需要从外部文件嵌入的数据，如 img 元素中的图像文件等，类似于“data:image/png; base64, iVBORwOKGgoAAAANSUhEUgAAAAoAAAAK...etc”。目前，大多数现代浏览器都支持该功能。

使用 toBlob()方法，可以把画布存储到 Blob 对象中，用以展示 canvas 上的图片；这个图片文件可以被缓存或保存到本地。具体用法如下：

```
void canvas.toBlob(callback, type, encoderOptions);
```

参数 callback 表示回调函数，当存储成功时调用，可获得一个单独的 Blob 对象参数。type 和 encoderOptions 参数与 toDataURL()方法相同。

【示例 1】下面示例将绘图输出到 data URL，效果如图 4.54 所示。

```
<canvas id="myCanvas" width="400" height="200"></canvas>
```



```
<script type="text/javascript">
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext('2d');
context.fillStyle = "rgb(0, 0, 255)";
context.fillRect(0, 0, canvas.width, canvas.height);
context.fillStyle = "rgb(255, 255, 0)";
context.fillRect(10, 20, 50, 50);
window.location =canvas.toDataURL("image/jpeg");
</script>
```





图 4.54　把图形输出到 data URL

【示例 2】下面示例在页面中添加一块画布、两个按钮，画布中显示绘制的几何图形，单击“保存图像”按钮，可以把绘制的图形另存到另一个页面中，单击“下载图像”按钮，可以把绘制的图形下载到本地，演示效果如图 4.55 所示。

具体代码解析请扫码学习。

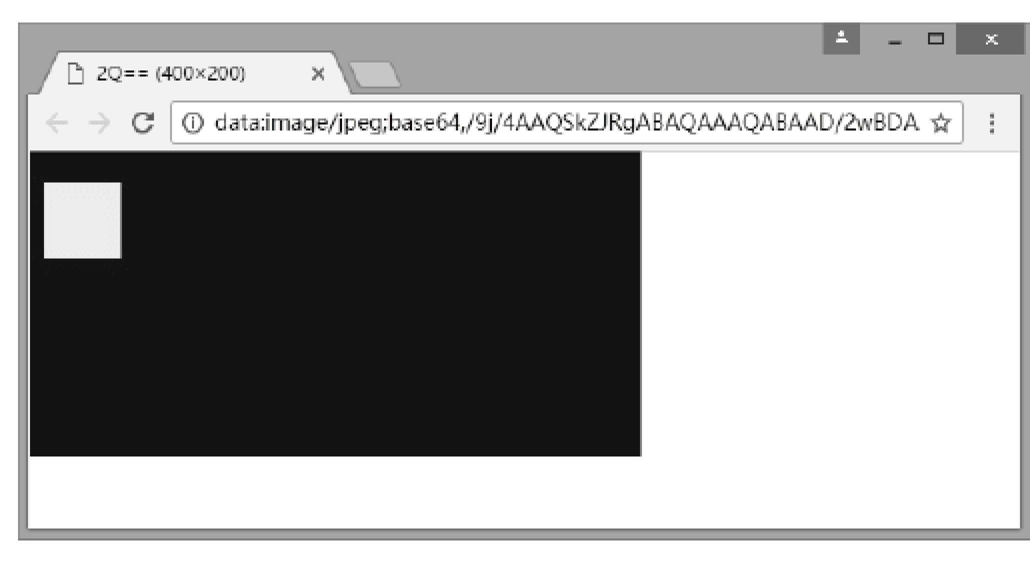


图 4.55　保存和下载图形



线上阅读

4.9　Path2D 对象

HTML Canvas 2D API 新增了一些新的功能。目前，Chrome 37+、Firefox 31+、Opera 23+、Safari 7+版本浏览器支持或部分支持以 Path2D 对象为核心的新功能。

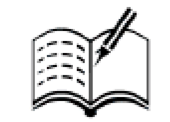
4.9.1　Canvas 2D API 新功能



视频讲解

在 MDN 上， Canvas 2D API 文档已经进行了大部分更新，以反映当前 canvas 标准和浏览器的执行状态。新增功能简单描述如下：





Note

1. 新增 Path2D 对象

新的 Path2D API 从 Firefox 31+开始支持，允许用户存储路径，简化了 canvas 绘制代码，并提升了运行速度。用户可以通过三种方式创建一个 Path2D 对象。

```
new Path2D();          //空的 path 对象
new Path2D(path);      //复制 path 对象
new Path2D(d);         //通过 SVG path 字符串创建 path 对象
```

第三种方式使用了 SVG 路径数据来构建，非常好用。用户现在也可以重复使用自己的 SVG 路径在 canvas 中直接来绘制同样的形状。例如：

```
var p = new Path2D("M10 10 h 80 v 80 h -80 Z");
```

在构建一个空的路径对象时，用户可以在 CanvasRenderingContext2D 环境中直接使用自己熟悉的常用的路径方法。例如：

```
//创建一个圆
var circle = new Path2D();
circle.arc(50, 50, 50, 0, 2 * Math.PI);
//在 context 上下文对象 ctx 中描绘边框
ctx.stroke(circle);
```

在 canvas 实际绘制路径时，提供一个可选的 Path2D 路径：

```
ctx.fill(path)
ctx.stroke(path)
ctx.clip(path)
ctx.isPointInPath(path)
ctx.isPointInStroke(path)
```

2. 点击区域

从 Firefox 32 开始，对点击区域（hit regions）的实验性支持被添加进来。用户需要设置 canvas.hitregions.enabled=true 来进行测试。

点击区域提供了一个更方便的方法来探测鼠标是否在一个特定区域。不再手动检查坐标，这对复杂的形状来说非常困难。定义点击区域方法如下：

```
CanvasRenderingContext2D.addHitRegion()        //在 canvas 中添加一个点击区域
CanvasRenderingContext2D.removeHitRegion()     //从 canvas 中移除带有 id 的点击区域
CanvasRenderingContext2D.clearHitRegions()     //从 canvas 中移除所有的点击区域
```

addHitRegion()方法可以在一个现有的路径或是一个 Path2D 路径中添加一个点击区域。MouseEvent 接口有一个扩展的 region 属性，用户可以用其来检查鼠标是否点击了区域。

3. 焦点环

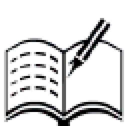
在 Firefox 32 中，drawFocusIfNeeded(element)可以无须属性切换自动支持。如果在<canvas>元素中提供的回退元素获得焦点，用户可以使用这个 API 在 canvas 中绘制一个焦点环。

如果回退元素获得焦点（如切换到一个包含 canvas 的页面），该元素在 canvas 中的像素表示/形状可以绘制一个焦点环来表示当前的焦点。

4. CSS/SVG 过滤器可用于 canvas

Firefox 35 开始支持 canvas 渲染内容的过滤器。语法和 CSS 过滤器属性相同。







5. 其他

☑ alpha 属性在 Firefox 30 中可用。
☑ 可以在样式中添加 transformations，Firefox 33+支持。
☑ 新增 ctx.resetTransform() 方法来重置变型，Firefox 36+支持。
☑ 最新的 canvas 规范包含了一些全新的 API，以及在不同浏览器中的执行情况。

4.9.2　使用 Path2D 对象

使用 Path2D 对象的各种方法可以绘制直线、矩形、圆形、椭圆以及曲线，具体说明如表 4.2 所示。这些方法中的各种参数的用法与图形上下文对象的同名方法用法相同。

表 4.2　Path2D 对象方法

方　　法	说　　明
path.moveTo(x,y)	将光标移动到指定坐标点
path.lineTo(x,y)	在当前坐标点与参数中指定的直线终点之间绘制一条直线
path.rect(x, y, width, height)	绘制矩形
path.arc(x, y, radius, startAngle, endAngle, anticlockwise);	绘制圆形或圆弧
path. ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle, anticlockwise);	绘制椭圆或椭圆形圆弧
path.arcTo(x1, y1, x2, y2, radius);	绘制圆形曲线或圆弧曲线
path.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y);	绘制贝塞尔曲线
path.quadraticCurveTo(cpx, cpy, x, y);	绘制二次样条曲线
path.closePath()	关闭路径

可以使用图形上下文对象的 fill()方法填充使用 Path2D 对象绘制的路径所形成的图形，或者使用图形上下文对象的 stroke()方法绘制使用 Path2D 对象绘制的路径所形成的图形轮廓，代码如下所示：

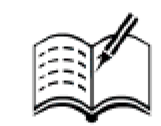
```
context.fill (path);
context.stroke(path);
```

【示例 1】下面示例使用 path.arc()方法在画布上绘制多个圆形。在浏览器中的预览效果如图 4.56 所示。

```
<canvas id="myCanvas" width="600" height="400"></canvas>
<script type="text/javascript">
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext('2d');
context.fillStyle = "#EEEEFF";
context.fillRect(0, 0, 600, 400);
var n = 0;
for(var i = 0; i < 10; i++){
    var path=new Path2D();
    path.arc(i * 25, i * 25, i * 10, 0, Math.PI * 2, true);
    path.closePath();
    context.fillStyle = 'rgba(255, 0, 0, 0.25)';
```



```
        context.fill(path);
    }
</script>
```



可以在 Path2D 对象的构造函数中使用一个参数，参数值为另外一个 Path2D 对象，这将该对象所代表的路径复制给新创建的 Path2D 对象。

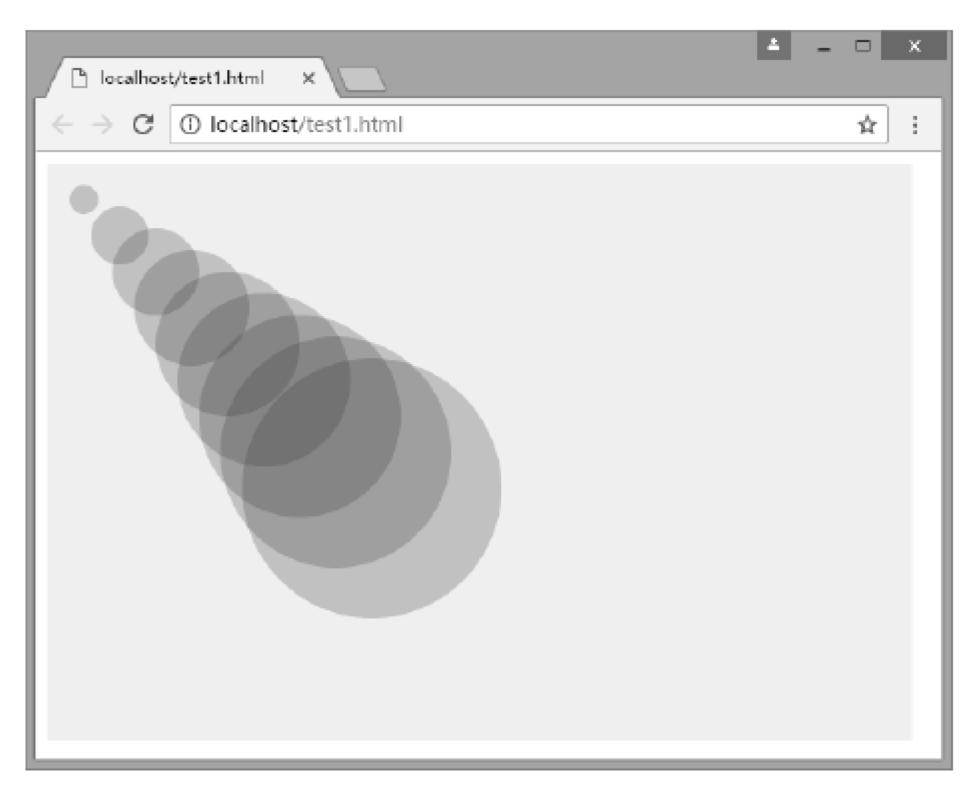


图 4.56　重复绘制圆形

【示例 2】下面示例首先创建一个 Path2D 对象，并使用该对象绘制一个矩形路径。然后，创建第二个 Path2D 对象，并将第一个 Path2D 对象所代表的路径复制给第二个 Path2D 对象。最后，再次使用第二个 Path2D 对象绘制一个圆形路径，使用图形上下文对象的 stroke()方法在画布中绘制第二个 Path2D 对象所形成的图形。在浏览器中的预览效果如图 4.57 所示。

```
<canvas id="myCanvas" width="300" height="200"></canvas>
<script type="text/javascript">
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext('2d');
var path1=new Path2D();
path1.rect(10,10,100,100);
var path2=new Path2D(path1);
path2.moveTo(220,60);
path2.arc(170,60,50,0,2*Math.PI);
context.stroke(path2);
</script>
```

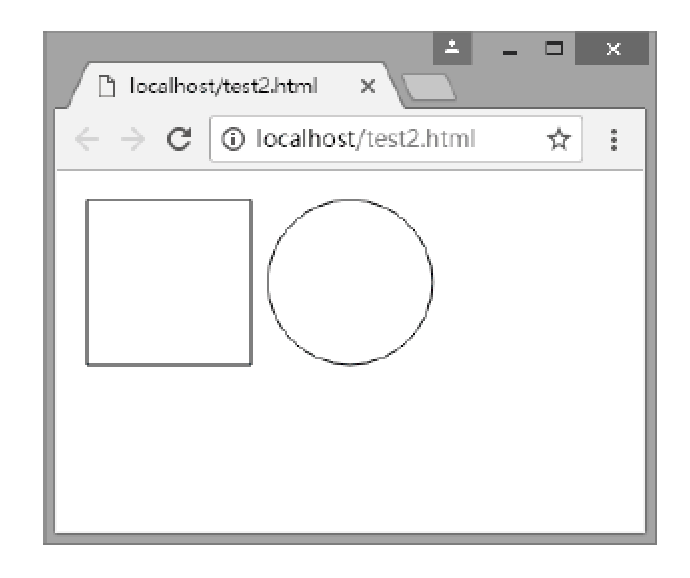
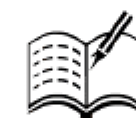


图 4.57　复制绘制的圆形

可以使用 Path2D 对象的 addPath()方法将一个 Path2D 对象所代表的路径添加到另一个 Path2D 对象所代表的路径中，方法如下所示：





```
path2.addPath(pathl);
```

在上面语法中，path2 与 path 1 均代表一个 Path2D 对象，该行代码表示将 path 1 对象所代表的路径添加到 path2 对象所代表的路径中。

【示例 3】下面示例修改示例 2 中的代码，如下所示。在浏览器中访问页面，显示效果如图 4.58 所示。

```
<canvas id="myCanvas" width="300" height="200"></canvas>
<script type="text/javascript">
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext('2d');
var path1=new Path2D();
path1.rect(10,10,100,100);
var path2=new Path2D();
path2.moveTo(220,60);
path2.arc(170,60,50,0,2*Math.PI);
path2.addPath(path1);
context.stroke(path2);
</script>
```

可以在 Path2D 对象的构造函数中使用一个代表了 SVG 路径的字符串，这表示使用该 Path2D 对象绘制该路径。

【示例 4】下面示例首先将绘制起点设置在（10, 10）处，然后横向绘制 80 个像素，纵向绘制 80 个像素，接着横向反向绘制 80 个像素，最后绘制到起点处。在浏览器中的预览效果如图 4.59 所示。

```
<canvas id="myCanvas" width="200" height="200"></canvas>
<script type="text/javascript">
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext('2d');
var path1=new Path2D("M10 10 h 80 v 80 h -80 Z ");
context.fill(path1);
</script>
```

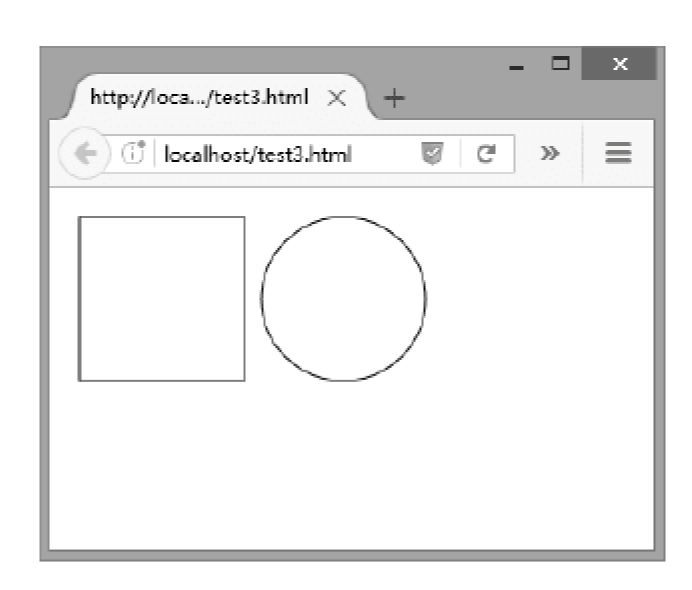


图 4.58　在 Firefox 中预览效果

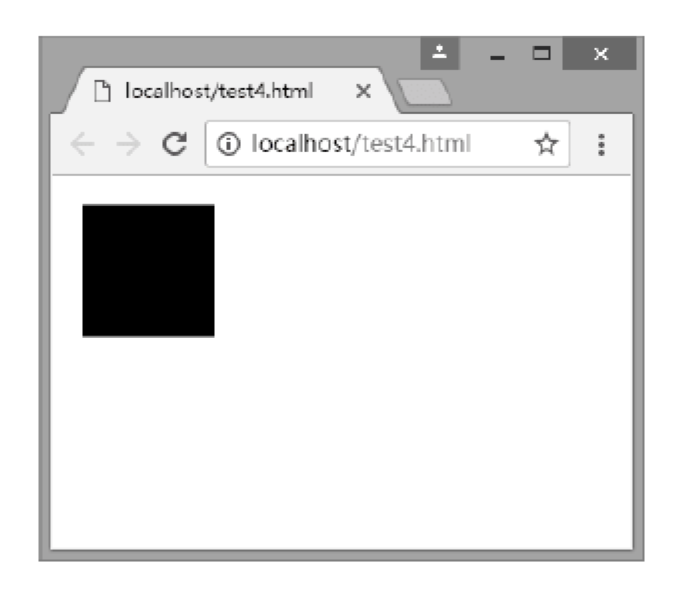


图 4.59　使用 SVG 路径绘制图形

4.10　案例实战

本节将结合案例介绍 Canvas API 高级应用。


视频讲解


Note

4.10.1 设计基本动画

设计动画的基本步骤：

第 1 步，清空 canvas。最简单的方法是使用 clearRect ()方法清空画布。

第 2 步，保存 canvas 状态。如果要改变 canvas 设置状态，如样式、变形等，又要在每画一帧之时都要重设原始状态，这时需要使用 save()方法先保存 canvas 设置状态。

第 3 步，绘制动画图形。这一步才是重绘动画帧。

第 4 步，恢复 canvas 状态，如果已经保存了 canvas 的状态，可以使用 restore()方法先恢复它，然后重绘下一帧。

有 3 种方法可以实现动画操控：

☑ setInterval(function, delay)：设定好间隔时间 delay 后，function 会定期执行。

☑ setTimeout(function, delay)：在设定好的时间 delay 之后，执行函数 function。

☑ requestAnimationFrame(callback)：告诉浏览器希望执行动画，并请求浏览器调用指定的 callback 函数在下一次重绘之前更新动画。requestAnimationFrame()函数不需要指定动画关键帧间隔的时间，浏览器自动设置。

【示例】下面示例在画布中绘制一个红色方块和一个圆形球，让它们重叠显示，然后使用一个变量从图形上下文的 globalCompositeOperation 属性的所有参数构成的数组中挑选一个参数来显示对应的图形组合效果，通过动画来循环显示所有参数的组合效果，演示如图 4.60 所示。

具体代码解析请扫码学习。


线上阅读

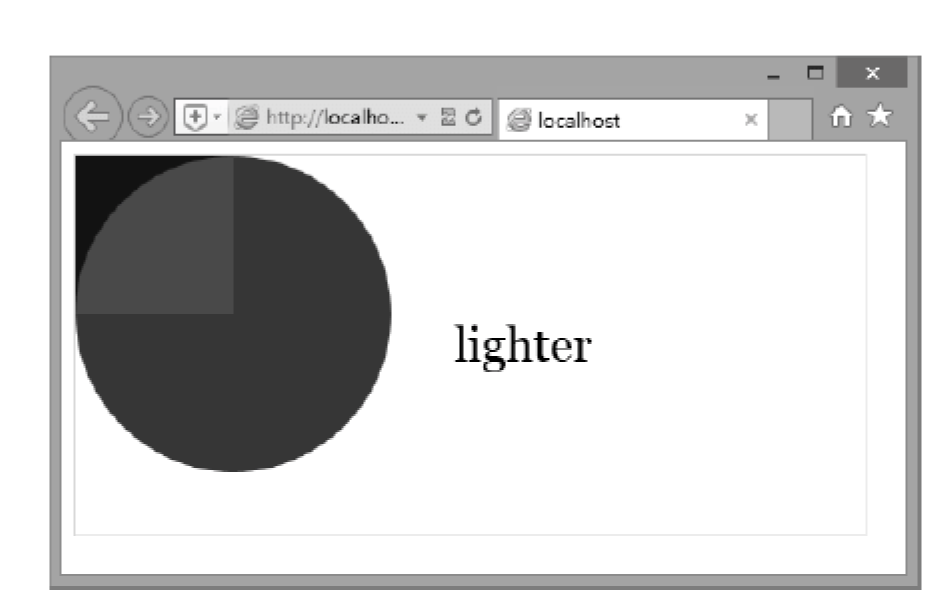


图 4.60 设计图形组合动画


视频讲解

4.10.2 颜色选择器

本例使用 getImageData()方法展示鼠标光标移动下的颜色，效果如图 4.61 所示。

具体代码解析请扫码学习。


线上阅读



图 4.61 设计图形组合动画





视 频 讲 解



Note

4.10.3　给图像去色

本例在 4.8.4 节示例基础上，设计通过按钮控制图像的色彩处理，当单击“反色图像”按钮时，让图像反色显示，当单击“灰色图像”按钮时，让图像以灰度图显示，效果如图 4.62 所示。



（a）反色图像



（b）灰色图像

图 4.62　设计图像色彩处理



线 上 阅 读

具体代码解析请扫码学习。

4.10.4　缩放图像和反锯齿处理



视 频 讲 解

在 drawImage()方法中， 通过缩放第二块画布的大小，可以实现图像的实时缩放显示，再利用画布环境的 imageSmoothingEnabled 属性，可以设置放大显示的图像是否以反锯齿（即平滑方式）显示。示例代码如下所示，演示效果如图 4.63 所示。



（a）以锯齿方式放大图像



（b）以平滑方式放大图像

图 4.63　设计缩放图像

线 上 阅 读

具体代码解析请扫码学习。

4.10.5　设计运动动画



视 频 讲 解

在上面示例中，我们初步掌握了基本动画的设计方法，本节我们将会对运动有更深的了解并学会添加一些符合物理的运动。设计效果如图 4.64 所示。





Note



线上阅读

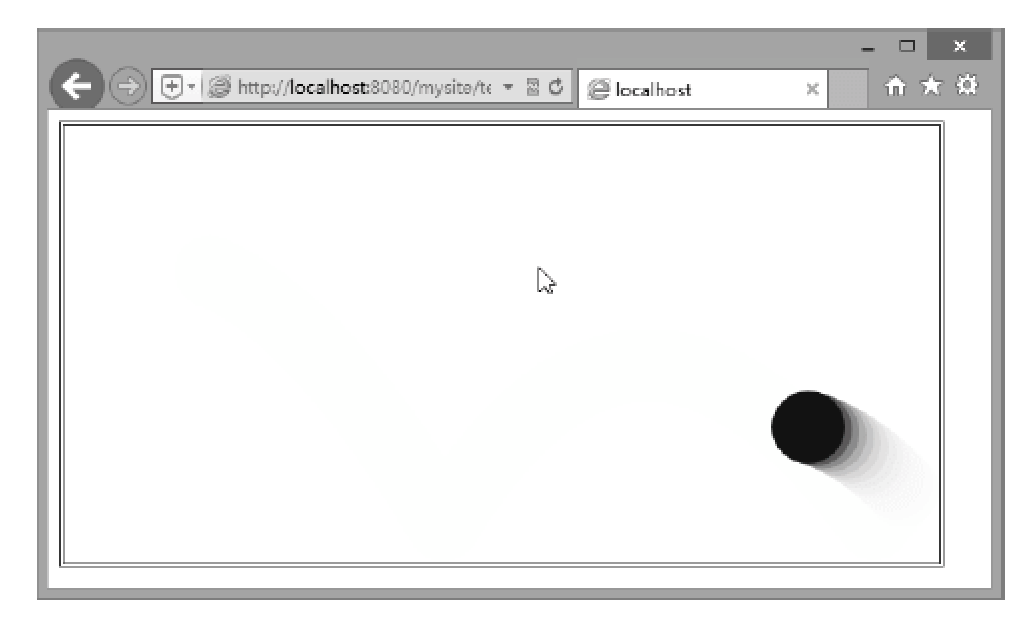


图 4.64　设计长尾效果

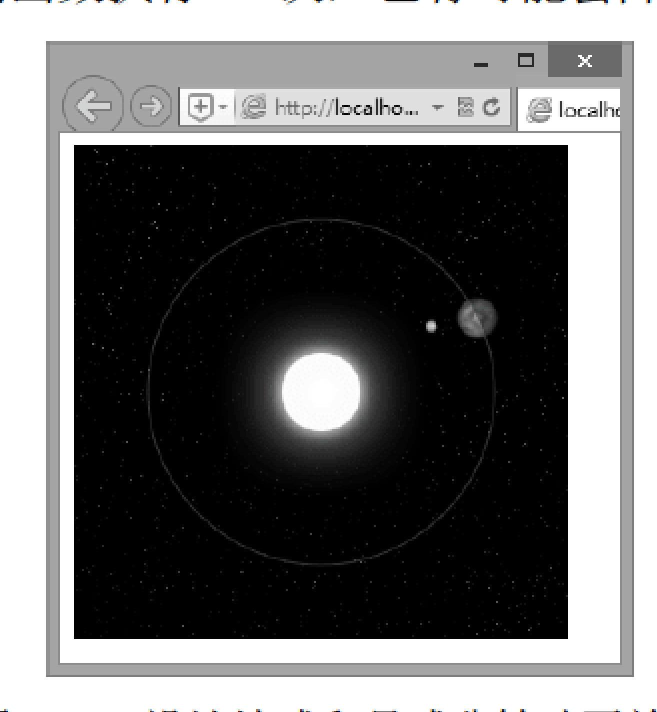
具体操作步骤请扫码学习。



视频讲解

4.10.6　设计地球和月球公转动画

本例采用 window.requestAnimationFrame()方法做一个小型的太阳系模拟动画，效果如图 4.65 所示。这个方法提供了更加平缓并更加有效率的方式来执行动画，当系统准备好了重绘条件的时候，才调用绘制动画帧。一般每秒钟回调函数执行 60 次，也有可能会降低。



线上阅读

图 4.65　设计地球和月球公转动画效果

具体代码解析请扫码学习。

4.11　在线练习

使用 HTML Canvas（画布）实现 2D 绘制图形、图像及其他动画效果。



在线练习



第5章 HTML5 音频和视频

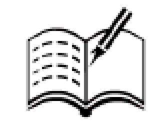
HTML5 新增了两个多媒体元素：audio 和 video，其中 audio 元素专门用来播放网络音频数据，而 video 元素专门用来播放网络视频或电影。另外，HTML5 规范了多媒体 API，允许用户通过 JavaScript 脚本操控多媒体对象。

【学习重点】

- 使用<audio>和<video>标签。
- 了解 audio 和 video 对象的属性、方法和事件。
- 能够使用 audio 和 video 设计音频和视频播放界面。



5.1　使用 HTML5 音频和视频



Note



视频讲解

目前，现代浏览器都支持 HTML5 的 audio 元素和 video 元素，如 IE 9.0+、Firefox 3.5+、Opera 10.5+、Chrome 3.0+、Safari 3.2+等。

5.1.1　使用<audio>

<audio>标签可以播放声音文件或音频流，支持 Ogg Vorbis、MP3、Wav 等音频格式，其用法如下。

```
<audio src="samplesong.mp3" controls="controls"></audio>
```

其中 src 属性用于指定要播放的声音文件，controls 属性用于设置是否显示工具条。<audio>标签可用的属性如表 5.1 所示。

表 5.1　<audio>标签支持属性

属　性	值	说　明
autoplay	autoplay	如果出现该属性，则音频在就绪后马上播放
controls	controls	如果出现该属性，则向用户显示控件，比如播放按钮
loop	loop	如果出现该属性，则每当音频结束时重新开始播放
preload	preload	如果出现该属性，则音频在页面加载时进行加载，并预备播放。如果使用 "autoplay"，则忽略该属性
src	url	要播放的音频的 URL



提示： 如果浏览器不支持<audio>标签，可以在<audio>与</audio>标识符之间嵌入替换的 HTML 字符串，这样旧的浏览器就可以显示这些信息。例如：

```
<audio src=" test.mp3" controls="controls">
您的浏览器不支持 audio 标签。
</audio>
```

替换内容可以是简单的提示信息，也可以是一些备用音频插件，或者是音频文件的链接等。

【示例 1】<audio>标签可以包含多个<source>标签，用来导入不同的音频文件，浏览器会自动选择第一个可以识别的格式进行播放。

```
<audio controls="controls">
    <source src="medias/test.ogg" type="audio/ogg">
    <source src="medias/test.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

以上代码在 Chrome 浏览器中的运行结果如图 5.1 所示，可以看到出现了一个比较简单的音频播放器，包含了播放、暂停、位置、时间显示、音量控制等常用控件按钮。



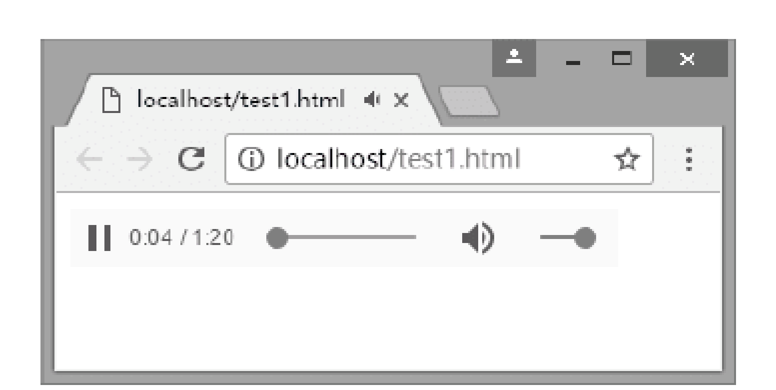


图 5.1　播放音频

【补充】

<source>标签可以为<video>和<audio>标签定义多媒体资源，它必须包裹在<video>或<audio>标识符内。<source>标签包含 3 个可用属性：

☑ media：定义媒体资源的类型。

☑ src：定义媒体文件的 URL。

☑ type：定义媒体资源的 MIME 类型。如果媒体类型与源文件不匹配，浏览器可能会拒绝播放。可以省略 type 属性，让浏览器自动检测编码方式。

为了兼容不同浏览器，一般使用多个<source>标签包含多种媒体资源。对于数据源，浏览器会按照声明顺序进行选择，如果支持的不止一种，那么浏览器会优先播放位置靠前的媒体资源。数据源列表的排放顺序应按照用户体验由高到低，或者服务器消耗由低到高列出。

【示例 2】下面示例演示了如何在页面中插入背景音乐：在<audio>标签中设置 autoplay 和 loop 属性，详细代码如下所示。

```
<audio autoplay loop>
    <source src="medias/test.ogg" type="audio/ogg">
    <source src="medias/test.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

5.1.2　使用<video>



视频讲解

<video>标签可以播放视频文件或视频流，支持 Ogg、MPEG 4、WebM 等视频格式，其用法如下。

```
<video src="samplemovie.mp4" controls="controls"></video>
```

其中 src 属性用于指定要播放的视频文件，controls 属性用于提供播放、暂停和音量控件。<video>标签可用的属性如表 5.2 所示。

表 5.2　<video>标签支持属性

属　性	值	描　述
autoplay	autoplay	如果出现该属性，则视频在就绪后马上播放
controls	controls	如果出现该属性，则向用户显示控件，如“播放”按钮
height	pixels	设置视频播放器的高度
loop	loop	如果出现该属性，则当媒介文件完成播放后再次开始播放
muted	muted	设置视频的音频输出应该被静音
poster	URL	设置视频下载时显示的图像，或者在用户单击“播放”按钮前显示的图像
preload	preload	如果出现该属性，则视频在页面加载时进行加载，并预备播放。如果使用 "autoplay"，则忽略该属性




Note

续表

属　性	值	描　述
src	url	要播放的视频的 URL
width	pixels	设置视频播放器的宽度

【补充】

HTML5 的<video>标签支持 3 种常用的视频格式，简单说明如下（浏览器支持情况：Safari 3+、Firefox 4+、Opera 10+、Chrome 3+、IE 9+等）。

☑ Ogg：带有 Theora 视频编码和 Vorbis 音频编码的 Ogg 文件。

☑ MPEG 4：带有 H.264 视频编码和 AAC 音频编码的 MPEG 4 文件。

☑ WebM：带有 VP8 视频编码和 Vorbis 音频编码的 WebM 文件。

提示： 如果浏览器不支持<video>标签，可以在<video>与</video>标识符之间嵌入替换的 HTML 字符串，这样旧的浏览器就可以显示这些信息。例如：

```
<video src=" test.mp4" controls="controls">
您的浏览器不支持 video 标签。
</video>
```

【示例 1】下面示例使用<video>标签在页面中嵌入一段视频，然后使用<source>标签链接不同的视频文件，浏览器会自己选择第一个可以识别的格式。

```
<video controls>
    <source src="medias/trailer.ogg" type="video/ogg">
    <source src="medias/trailer.mp4" type="video/mp4">
您的浏览器不支持 video 标签。
</video >
```

以上代码在 Chrome 浏览器中的运行结果如图 5.2 所示，当鼠标经过播放画面，可以看到出现一个比较简单的视频播放控制条，包含了播放、暂停、位置、时间显示、音量控制等常用控件。

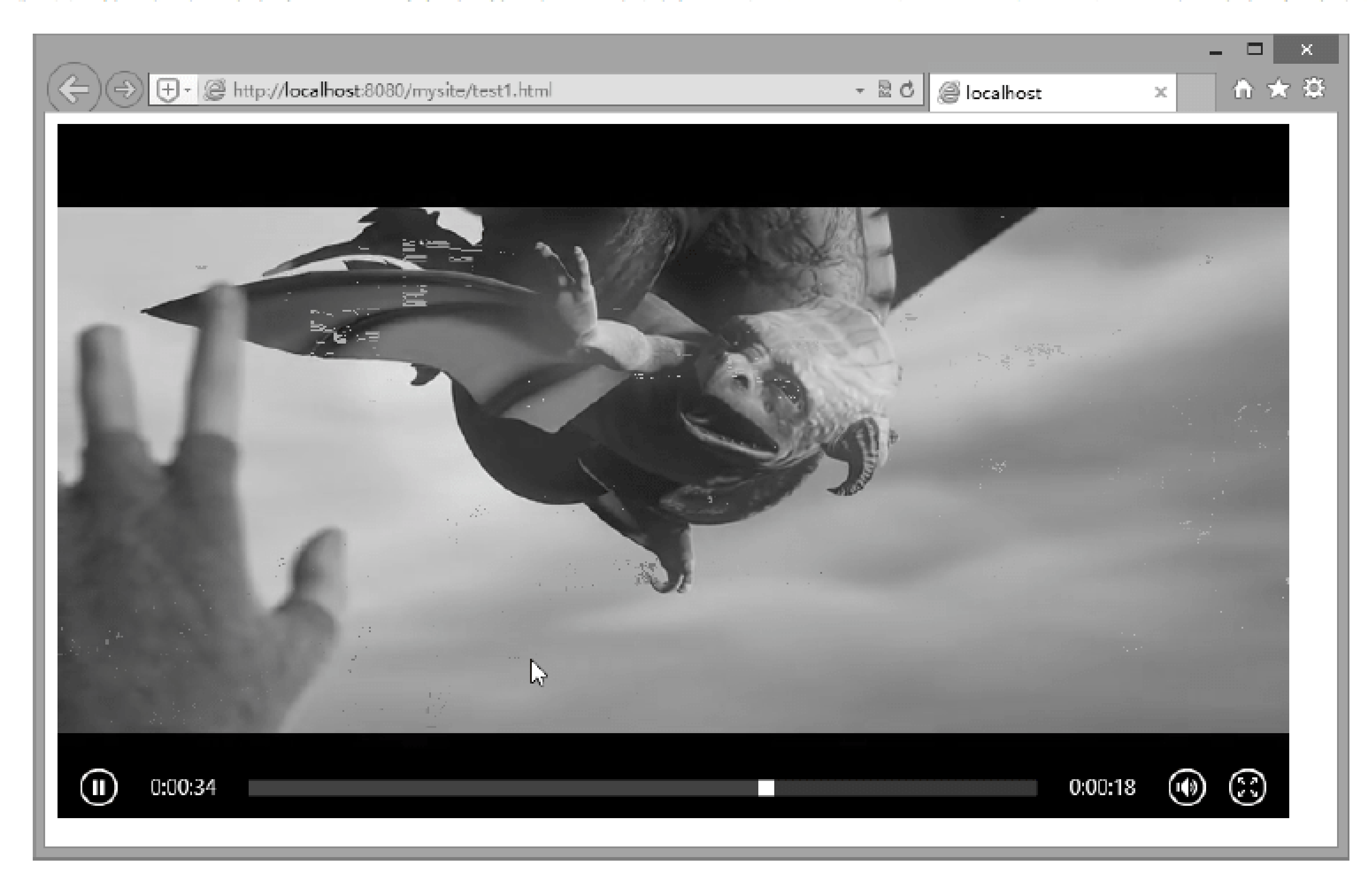
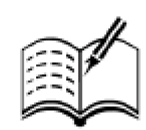


图 5.2　播放视频





为<video>标签设置 controls 属性，可以在页面上以默认方式进行播放控制。如果不设置 controls 属性，那么在播放的时候就不会显示控制条界面。

【示例 2】通过设置 autoplay 属性，不需要播放控制条，音频或视频文件就会在加载完成后自动播放。

```
<video autoplay>
    <source src="medias/trailer.ogg" type="video/ogg">
    <source src="medias/trailer.mp4" type="video/mp4">
您的浏览器不支持 video 标签。
</video >
```

也可以使用 JavaScript 脚本控制媒体播放，简单说明如下：

☑ load()：可以加载音频或者视频文件。

☑ play()：可以加载并播放音频或视频文件，除非已经暂停，否则默认从开头播放。

☑ pause()：暂停处于播放状态的音频或视频文件。

☑ canPlayType(type)：检测 video 元素是否支持给定 MIME 类型的文件。

【示例 3】下面示例演示如何通过移动鼠标来触发视频的 play 和 pause 功能。设计当用户移动鼠标到视频界面上时，播放视频，如果移出鼠标，则暂停视频播放。

```
<video id="movies" onmouseover="this.play()" onmouseout="this.pause()" autobuffer="true"
    width="400px" height="300px">
    <source src="medias/trailer.ogv" type='video/ogg; codecs="theora, vorbis"'>
    <source src="medias/trailer.mp4" type='video/mp4'>
</video>
```

上面代码在浏览器中预览，显示效果如图 5.3 所示。

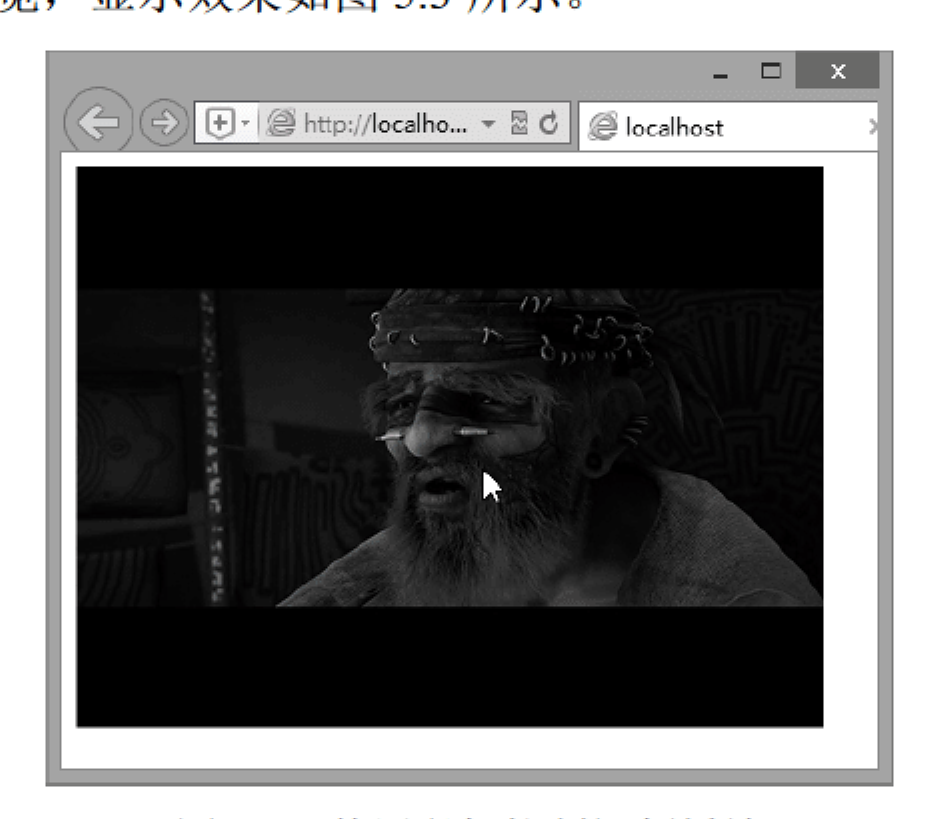


图 5.3　使用鼠标控制视频播放

5.1.3　设置属性

audio 和 video 元素拥有相同的脚本属性，下面对这些属性进行简单介绍。

☑ autobuffer 属性

可读写属性。使用该属性可以使得 audio 或 video 元素实现自动缓冲，默认值为 false，即 audio 或 video 元素默认情况下并不自动缓冲。如果值为 true，则自动缓冲，但并不播放。如果使用了 autoplay 属性，则 autobuffer 属性会被忽略。其用法见下面的示例。

```
<audio controls="controls" autobuffer="true">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ autoplay 属性

可读写属性。使用该属性可以实现页面加载后音频一旦就绪即开始自动播放。使用 autoplay 属性相比使用脚本控制音频或视频播放来得简便，其值也可以设置为 true 或 false。如果值为 true 或 autoplay，则当音频或视频缓冲到足够多时即开始播放。其用法见下面的示例。

```
<audio controls="controls" autoplay="autoplay">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ buffered 属性

只读属性，用于返回一个 TimeRanges 对象，确认浏览器已经缓存媒体文件。

☑ controls 属性

可读写属性。该属性为布尔值，可以为媒体文件提供用于播放的控制条，包含播放、暂停、定位、时间显示、音量控制、全屏切换等常用控件，其用法如下。在将来的标准中，有望在播放控件中看到字幕和音轨。开发人员如果不希望使用浏览器默认的控制条，也可以使用脚本来自定义控制条。其用法见下面的示例。

```
<audio controls="controls">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ currentSrc 属性

只读属性，无默认值。用于返回媒体数据的 URL 地址，如果媒体 URL 地址未指定，则返回一个空字符串。

☑ currentTime 属性

可读写属性，无默认值。用于获取或设置当前播放位置，返回值为时间，单位为秒。

☑ defaultPlaybackRate

可读写属性，无默认值。用于获取或设置当前播放速率，前提是用户没有使用快进或快退控件。

☑ duration 属性

只读属性，无默认值。用于获取当前媒体的持续时间，返回值为时间，单位为秒。

☑ ended 属性

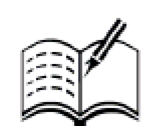
只读属性，无默认值。用于返回一个布尔值，以获悉媒体是否播放结束。

☑ error 属性

只读属性，无默认值。用于返回一个 MediaError 对象以表明当前的错误状态，如果没有出现错误，则返回 null。错误状态共有 4 个可能值，分别为：

- MEDIA_ERR_ABORTED（数字值为 1）：媒体资源获取异常——媒体数据的下载过程因用户操作而终止。





- ➢ MEDIA_ERR_NETWORK（数字值为 2）：网络错误—— 在媒体数据已经就绪时用户停止了媒体下载资源的过程。
- ➢ MEDIA_ERR_DECODE（数字值为 3）：媒体解码错误—— 在媒体数据已经就绪时解码过程中出现了错误。
- ➢ MEDIA_ERR_SRC_NOT_SUPPORTED（数字值为 4）：媒体格式不被支持。

☑ initialTime 属性

只读属性，无默认值。用于获取最早的可用于回放的位置，返回值为时间，单位为秒。

☑ loop 属性

可读写属性。用于获取或设置当媒体文件播放结束时是否再重新开始播放。其使用方法如下所示。

```
<audio controls="controls" loop="loop">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ muted 属性

可读写属性，无默认值。muted 属性为布尔值，用于获取或设置当前媒体播放是否开启静音，true 为开启静音，false 为未开启静音。如果对其赋值，则可以设置播放时是否静音。

☑ networkState 属性

只读属性。用于返回媒体的网络状态，共有 4 个可能值。

- ➢ NETWORK_EMPTY（数字值为 0）：元素尚未初始化。
- ➢ NETWORK_IDLE（数字值为 1）：加载完成，网络空闲。
- ➢ NETWORK_LOADING（数字值为 2）：媒体数据加载中。
- ➢ NETWORK_NO_SOURCE（数字值为 3）：因为不存在支持的编码格式，加载失败。

☑ paused 属性

只读属性，无默认值。用于返回一个布尔值，表示媒体是否暂停播放，true 表示暂停，false 表示正在播放。

☑ playbackRate 属性

可读写属性，无默认值。用于读取或设置媒体资源播放的当前速率。

☑ played 属性

只读属性，无默认值。用于返回一个 TimeRanges 对象，标明媒体资源在浏览器中已播放的时间范围。TimeRanges 对象的 length 属性为已播放部分的时间段，该对象有两个方法，end 方法用于返回已播放时间段的结束时间，start 方法用于返回已播放时间段的开始时间，其用法见下面的示例。

```
var ranges = document.getElementById('myVideo').played;
for (var i=0; i<ranges.length; i++)
    var start = ranges.start(i);
    var end = ranges.end(i);
    alert("从" + start +"开始播放到" + end+"结束。");
}
```

☑ preload 属性

可读写属性，无默认值。用于定义视频是否预加载，该属性有 3 个可选值：none、metadata 和 auto。如果不用该属性，则默认为 auto，分别介绍如下。

- ➢ none：不进行预加载。当页面制作人员认为用户不希望直接观看此视频，或者减少 HTTP



请求时可以设置该项。

- metadata：部分预加载。使用此属性值，代表页面制作者认为用户不期望直接观看此视频，但为用户提供一些元数据（包括尺寸、第一帧、曲目列表、持续时间等）。
- auto：全部预加载。



Note

preload 属性的用法如下所示。

```
<video src="samplemovie.mp4" preload="auto">
</video>
```

☑ readyState 属性

只读属性，无默认值。用于返回媒体当前播放位置的就绪状态，共有以下 5 个可能值。

- HAVE_NOTHING（数字值为 0）：当前播放位置没有有效的媒体资源。
- HAVE_METADATA（数字值为 1）：媒体资源确认存在且加载中，但当前位置没有能够加载到有效的媒体数据以进行播放。
- HAVE_CURRENT_DATA（数字值为 2）：已获取到当前播放数据，但没有足够的数据进行播放。
- HAVE_FUTURE_DATA（数字值为 3）：在当前位置已获取到后续播放媒体数据，可以进行播放。
- HAVE_ENOUGH_DATA（数字值为 4）：媒体数据可以进行播放，且浏览器确认媒体数据正以某一种速率进行加载并有足够的后续数据以继续进行播放，而不会使浏览器的播放进度赶上加载数据的末端。

☑ seekable 属性

只读属性，无默认值。用于返回一个 TimeRanges 对象，表明可以对当前媒体资源进行请求。

☑ seeking 属性

只读属性，无默认值。用于返回一个布尔值，表示浏览器是否正在请求某一播放位置的媒体数据，ture 表示浏览器正在请求数据，而 false 表示浏览器已经停止请求数据。

☑ src 属性

可读写属性，无默认值。用于指定媒体资源的 URL 地址，与<img>标签类似，可与 poster 属性连用。poster 属性用于指定一张替换图片，如果当前媒体数据无效，则显示该图片。其用法如下所示。

```
<video src="http://www.lidongbo.com/samplemovie.mp4" poster=" http://www.lidongbo.com/samplemovie.png">
</video>
```

☑ volume 属性

可读写属性，无默认值。用于获取或设置媒体资源的播放音量，范围为 0.0～1.0，0.0 为静音，1.0 为最大音量。注意音量大小并非是线性变化的，如果同时使用了 muted 属性，则此属性会被忽略。

5.1.4 设置方法



视 频 讲 解

audio 和 video 元素拥有相同的脚本方法，下面简单介绍这些方法。

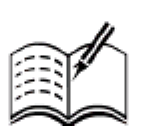
☑ canPlayType()方法

用于返回一个字符串以表明客户端是否能够播放指定的媒体类型，其用法如下。

```
var canPlay = media.canPlayType(type)
```

其中 media 指页面中的 audio 或 video 元素，参数 type 为客户端浏览器能够播放的媒体类型。该





方法返回以下可能值之一。

- ➢ probably：表示浏览器确定支持此媒体类型。
- ➢ maybe：表示浏览器可能支持此媒体类型。
- ➢ 空字符串：表示浏览器不支持此媒体类型。

☑ load()方法

用于重置媒体元素并重新载入媒体，不返回任何值，该方法可中止任何正在进行的任务或事件。元素的 playbackRate 属性值会被强行设为 defaultPlaybackRate 属性的值，而且元素的 error 值会被强行设置为 null。

【示例】在下面示例中，通过单击按钮可以重新载入另一个新的视频。

```
<video controls>
    <source src="medias/video.ogv" type='video/ogg'>
    <source src="medias/video.mp4" type='video/mp4'>
    您的浏览器不支持视频播放。
</video>
<input type="button" value="载入新的视频"  onclick="loadNewVideo()">
<script>
function loadNewVideo() {
    var video = document.getElementsByTagName('video')[0];
    var sources = video.getElementsByTagName('source');
    sources[0].src = 'medias/video2.ogv';
    sources[1].src = 'medias/video2.mp4';
    video.load(); //用 load 方法载入新的视频
}
</script>
```

☑ pause()方法

用于暂停媒体的播放，并将元素的 paused 属性的值强行设置为 true。

☑ play()方法

用于播放媒体，并将元素的 paused 属性的值强行设置为 false。

5.1.5　设置事件



audio 和 video 元素支持 HTML5 的媒体事件，详细说明如表 5.3 所示。使用 JavaScript 脚本可以捕捉这些事件并对其进行处理。处理这些事件一般有下面两种方式。

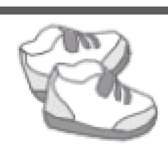
☑ 一种是使用 addEventListener()方法监听，其用法如下：

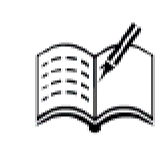
```
addEventListener("事件类型",处理函数,处理方式)
```

☑ 另一种是直接赋值，即获取事件句柄的方法。例如，video.onplay=begin_playing，其中 begin_playing 为处理函数。

表 5.3　音频与视频相关事件

事　　件	描　　述
abort	浏览器在完全加载媒体数据之前中止获取媒体数据
canplay	浏览器能够开始播放媒体数据，但估计以当前速率播放不能直接将媒体播放完，即可能因播放期间需要缓冲而停止





Note

续表

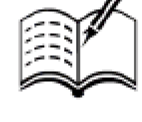
事　件	描　述
canplaythrough	浏览器以当前速率可以直接播放完整个媒体资源，在此期间不需要缓冲
durationchange	媒体长度（duration 属性）改变
emptied	媒体资源元素突然为空时，可能是网络错误或加载错误等
ended	媒体播放已抵达结尾
error	在元素加载期间发生错误
loadeddata	已经加载当前播放位置的媒体数据
loadedmetadata	浏览器已经获取媒体元素的持续时间和尺寸
loadstart	浏览器开始加载媒体数据
pause	媒体数据暂停播放
play	媒体数据将要开始播放
playing	媒体数据已经开始播放
progress	浏览器正在获取媒体数据
ratechange	媒体数据的默认播放速率（defaultPlaybackRate 属性）改变或播放速率（playbackRate 属性）改变
readystatechange	就绪状态（ready-state）改变
seeked	浏览器停止请求数据，媒体元素的定位属性不再为真（seeking 属性值为 false）且定位已结束
seeking	浏览器正在请求数据，媒体元素的定位属性为真（seeking 属性值为 true）且定位已开始
stalled	浏览器获取媒体数据过程中出现异常
suspend	浏览器非主动获取媒体数据，但在取回整个媒体文件之前中止
timeupdate	媒体当前播放位置（currentTime 属性）发生改变
volumechange	媒体音量（volume 属性）改变或静音（muted 属性）
waiting	媒体已停止播放但打算继续播放

【示例】下面示例使用 play()和 pause()方法控制视频的播放和暂停播放，使用 ended 事件监听视频播放是否完毕，使用 error 事件监听播放过程中发生的各种异常，并及时进行提示。

```
<body onload="init()">
<video id="video1" autoplay oncanplay="startVideo()" onended="stopTimeline()" autobuffer="true"
    width="400px" height="300px">
    <source src="medias/volcano.ogv" type='video/ogg'>
    <source src="medias/volcano.mp4" type='video/mp4'>
</video><br>
<button onclick="play()">播放</button>
<button onclick="pause()">暂停</button>
<script type="text/javascript">
```



```
var video;
function init(){
    video = document.getElementById("video1");
    //监听视频播放结束事件
    video.addEventListener("ended", function(){
        alert("播放结束。");
    }, true);
    //发生错误
    video.addEventListener("error",function(){
        switch (video.error.code){
            case MediaError.MEDIA_ERROR_ABORTED:
                alert("视频的下载过程被中止。");
                break;
            case MediaError.MEDIA_ERROR_NETWORK:
                alert("网络发生故障，视频的下载过程被中止。");
                break;
            case MediaError.MEDIA_ERROR_DECODE:
                alert("解码失败。");
                break;
            case MediaError.MEDIA_ERROR_SRC_NOT_SUPPORTED:
                alert("媒体资源不可用或媒体格式不被支持。");
                break;
            default:
                alert("发生未知错误。");
        }
    },false);
}
function play(){//播放视频
    video.play();
}
function pause(){ //暂停播放
    video.pause();
}
</script>
</body>
```



Note

5.2　案 例 实 战

本节将通过多个案例练习如何灵活使用 JavaScript 脚本控制 HTML5 多媒体播放。

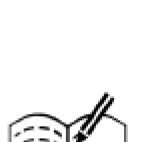
5.2.1　设计音乐播放器



视频讲解

如果需要在页面上播放一段音频，同时又不想被默认的控制界面影响显示效果，则可创建一个隐藏的 audio 元素，即不设置 controls 属性，或将其设置为 false，然后用自定义控制界面控制音频的播放。本例主要代码如下，演示效果如图 5.4 所示。





Note





线 上 阅 读

图 5.4　用脚本控制音乐播放

具体代码解析请扫码学习。



视 频 讲 解

5.2.2　获取播放进度

在播放过程中会经常触发 timeupdate 事件，通过该事件可以获取当前播放位置的变化。下面示例捕捉 timeupdate 事件来显示当前的播放进度。在 Chrome 浏览器中预览，可以在文本框中输入一个要播放的视频路径，或者自动播放默认视频，这时可以看到按钮右侧显示当前视频总长度，以及播放进度，效果如图 5.5 所示。





线 上 阅 读

图 5.5　显示播放进度

具体代码解析请扫码学习。

视 频 讲 解

5.2.3　设计视频播放器

本例将设计一个视频播放器，用到 HTML5 提供的 video 元素、以及 HTML5 提供的多媒体 API 的扩展，示例演示效果如图 5.6 所示。





Note



图 5.6　设计视频播放器

使用 JavaScript 控制播放控件的行为（自定义播放控件），实现如下功能：

☑　利用 HTML+CSS 制作一个自己的播放控件条，然后定位到视频最下方。

☑　视频加载 loading 效果。

☑　播放、暂停。

☑　总时长和当前播放时长显示。

☑　播放进度条。

☑　全屏显示。

具体操作步骤请扫码学习。



线 上 阅 读



视 频 讲 解

5.2.4　视频自动截图

本示例将演示如何抓取 video 元素中的帧画面并显示在动态 canvas 上。当视频播放时，定期从视频中抓取图像帧并绘制到旁边的 canvas 上，当用户单击 canvas 上显示的任何一帧时，所播放的视频会跳转到相应的时间点。演示效果如图 5.7 所示。

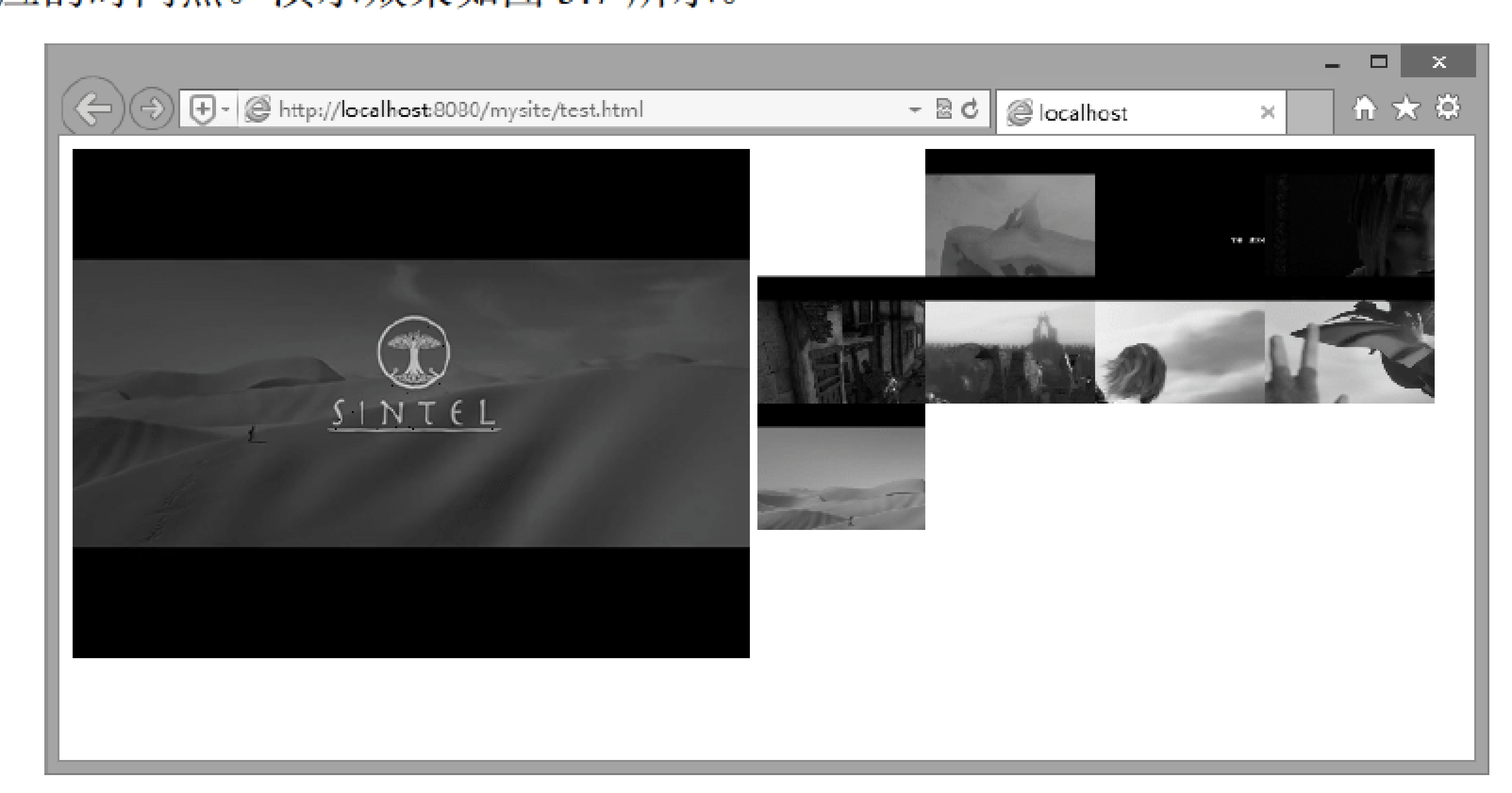


图 5.7　查看视频帧画面

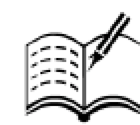


线 上 阅 读

具体操作步骤请扫码学习。



视频讲解



Note

5.2.5　视频同步字幕

HTML5 新增 track 元素，用于为 video 元素播放的视频或使用 audio 元素播放的音频添加字幕、标题等文字信息。track 元素允许用户沿着 audio 元素所使用的音频文件中的时间轴，或者 video 元素所使用的视频文件中的时间轴指定时间同步的文字资源。

目前，Chrome 18+、Firefox 28+、IE 10+、Opera 12+和 Safari 6+以上版本浏览器提供对 track 元素的支持，不包括 Firefox 30。

track 是一个空元素，其开始标签与结束标签之间并不包含任何内容，必须被书写在 video 或 audio 元素内部。如果使用 source 元素，则 track 元素必须被书写在 source 元素之后。用法如下所示：

```
<video width="320" height="240" controls="controls">
    <source src="forrest_gump.mp4" type="video/mp4" />
    <source src="forrest_gump.ogg" type="video/ogg" />
    <track kind="subtitles" src="subs_chi.srt" srclang="zh" label="Chinese">
    <track kind="subtitles" src="subs_eng.srt" srclang="en" label="English">
</video>
```

【示例】下面示例使用 video 元素播放一段视频，同时使用 track 元素在视频中显示字幕信息，演示效果如图 5.8 所示。

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<video src="medias/test.webm" controls>
    <track kind="subtitles" src="medias/test.vtt" default></track>
    您的浏览器不支持 video 元素
</video>
</body>
</html>
```

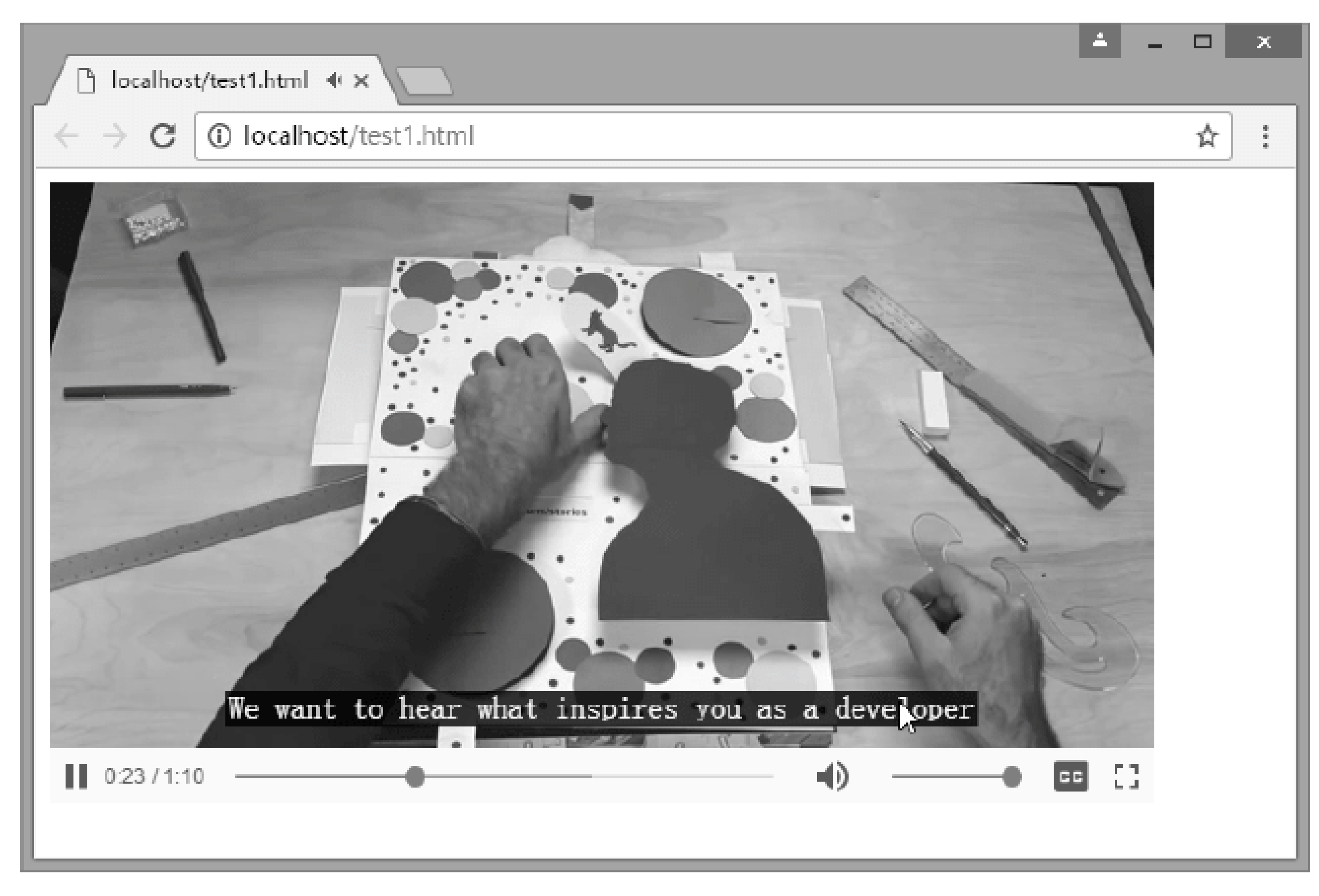
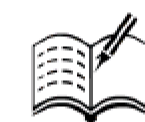


图 5.8　为视频添加字幕





Note

在 HTML5 中，track 元素包含几个特殊用途的属性，说明如表 5.4 所示。

表 5.4　track 元素属性

属　性	值	说　明
default	default	规定该轨道是默认的，假如没有选择任何轨道。例如： <track kind="subtitles" default src="chisubs.srt" srclang="zh">
kind	captions chapters descriptions metadata subtitles	表示轨道属于什么文本类型。例如： <video width="320" height="240" controls="controls"> <source src="forrest_gump.mp4" type="video/mp4" /> <track kind="subtitles" src="subschi.srt" srclang="zh" label="Chinese"> <track kind="subtitles" src="subseng.srt" srclang="en" label="English"> </video>
label	label	轨道的标签或标题。例如： <track kind="subtitles" label="Chinese subtitles" src="subschi.srt" srclang="zh" label="Chinese">
src	url	轨道的 URL
srclang	language_code	轨道的语言，若 kind 属性值是"subtitles"，则该属性是必需的。例如： <track kind="subtitles" src="subschi.srt" srclang="zh" label="Chinese">

其中 kind 属性的取值说明如下所示。

☑ captions：该轨道定义将在播放器中显示的简短说明。

☑ chapters：该轨道定义章节，用于导航媒介资源。

☑ descriptions：该轨道定义描述，用于通过音频描述媒介的内容（假如内容不可播放或不可见）。

☑ metadata：该轨道定义脚本使用的内容。

☑ subtitles：该轨道定义字幕，用于在视频中显示字幕。

【拓展】

网络视频文本轨道，简称为 WebVTT，是一种用于标记文本轨道的文件格式。它与 HTML5 的 track 元素相结合，可给音频、视频等媒体添加字幕、标题和其他描述信息，并同步显示。

1. 文件格式

WebVTT 文件是一个以 UTF-8 为编码，以.vtt 为文件扩展名的文本文件。

注意：如果要在服务器上使用 WebVTT 文件，可能需要显性定义其内容类型，例如，在 Apache 服务器的 .htaccess 文件中加入：

```
<Files mysubtitle.vtt>
    ForceType text/vtt;charset=utf-8
</Files>
```

WebVTT 文件的头部按如下顺序定义：

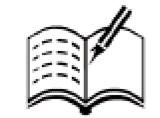
（1）可选的字节顺序标记（BOM）。



（2）字符串 WEBVTT。

（3）一个空格（Space）或者制表符（Tab），后面接任意非回车换行的元素。

（4）两个或两个以上的“WEBVTT 行结束符”：回车\r，换行\n，或者同时回车换行\r\n。



例如：

```
WEBVTT

Cue-1
00:00:15.000 --> 00:00:18.000
At the left we can see...
```

2. WebVTT Cues

WebVTT 文件包含一个或多个 WebVTT Cues，各个 WebVTT Cues 之间用两个或多个 WebVTT 行结束符分隔开来。

WebVTT Cue 允许用户指定特定时间戳范围内的文字（如字幕），也可以给 WebVTT Cue 指定一个唯一的标识符，标识符由简单字符串构成，不包含“-->”，也不包含任何的 WebVTT 行结束符。每一个提示采用以下格式：

```
[idstring]
[hh:]mm:ss.msmsms --> [hh:]mm:ss.msmsms
Text string
```

标识符是可选项，建议加入，因为它能够帮助组织文件，也方便脚本操控。

时间戳遵循标准格式：小时部分[hh:]是可选的，毫秒和秒用一个点（.）分离，而不是冒号（:）。时间戳范围的后者必须大于前者。对于不同的 Cues，时间戳可以重叠，但在单个 Cue 中，不能有字符串“-->”，或两个连续的行结束符。

时间范围后的文字可以是单行或者多行。特定的时间范围之后的任何文本都与该时间范围匹配，直到一个新的 Cue 出现或文件结束。例如：

```
Cue-8
00:00:52.000 --> 00:00:54.000
I don’t think so. You?

Cue-9
00:00:55.167 --> 00:00:57.042
I’m Ok.
```

3. WebVTT Cue 设置

在时间范围值后面可以设置 Cue：

```
[idstring]
[hh:]mm:ss.msmsms --> [hh:]mm:ss.msmsms [cue settings]
Text string
```

Cue 设置能够定义文本的位置和对齐方式，设置选项说明如表 5.5 所示。



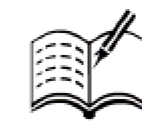


表 5.5　Cue 设置选项

设　　置	值	说　　明
vertical	rl || lr	将文本纵向向左对齐（lr）或向右对齐（rl）（如：日文的字幕）
line	[-][0 or more]	行位置，负数从框底部数起，正数从顶部数起
	[0-100]%	百分数意味着离框顶部的位置
position	[0-100]%	百分数意味着文字开始时离框左边的位置（如：英文字幕）
size	[0-100]%	百分数意味着 cue 框的大小是整体框架宽度的百分比
align	start || middle || end	指定 cue 中文本的对齐方式

注意：如果没有设置 Cue 选项，默认位置是底部居中。例如：

```
Cue-8
00:00:52.000 --> 00:00:54.000 align:start size:15%
I don't think so. You?

Cue-9
00:00:55.167 --> 00:00:57.042 align:end line:10%
I'm Ok.
```

在上面示例代码中，Cue-8 将靠左对齐，文本框大小为 15%，而 Cue-9 靠右对齐，纵向位置距离框顶部 10%。

4. WebVTT Cue 内联样式

用户可以使用 WebVTT Cue 内联样式来给 Cue 文本添加样式。这些内联样式类似于 HTML 元素，可以用来添加语义及样式。可用的内联样式说明如下：

- ☑ c：用 c 定义（CSS）类。例如，<c.className>Cue text</c>。
- ☑ i：斜体字。
- ☑ b：粗体字。
- ☑ u：添加下划线。
- ☑ ruby：定义类似于 HTML5 的<ruby>元素。在这样的内联样式中，允许出现一个或多个<rt>元素。
- ☑ v：指定声音标签。例如，<v Ian>This is useful for adding subtitles</v>。注意此声音标签不会显示，它只是作为一个样式标记。

例如：

```
Cue-8
00:00:52.000 --> 00:00:54.000 align:start size:15%
<v Emo>I don't think so. <c.question>You?</c></v>

Cue-9
00:00:55.167 --> 00:00:57.042 align:end line:10%
<v Proog>I'm Ok.</v>
```

上面示例给 Cue 文本添加两种不同的声音标签：Emo 和 Proog。另外，一个 question 的 CSS 类被指定，可以按惯常方法在 CSS 链接文件，或 HTML 页面里为其指定样式。





注意：要给 Cue 文本添加 CSS 样式，需要用一个特定的伪选择元素，例如：

```
video::cue(v[voice="Emo"]) { color:lime }
```

给 Cue 文本添加时间戳也是可能的，表示在不同的时间，不同的内联样式出现，例如：

```
Cue-8
00:00:52.000 --> 00:00:54.000
<c>I don't think so.</c> <00:00:53.500><c>You?</c>
```

虽然所有文本依旧在同一时间显示，不过在支持的浏览器中，可以用:past 和:future 伪类为其显示不同样式。例如：

```
video::cue(c:past) { color:yellow }
```

视频讲解

5.2.6　使用 HTML5 Web Audio API 增加声音

HTML5 中有关音频处理和播放的 API 包括两个：Web Audio 和 audio 元素，Web Audio 主要用来对音频数据添加音效，如过滤掉音频数据中的杂音让声音听起来更加圆润，也可以动态设置倾听者相对于音源的位置，让每个位置听起来的效果不一样，这在游戏中是经常需要的。

除此之外，Web Audio 提供了与其他元素的互操作性，如可以把 audio 元素中输出的音频数据导入 Web Audio 中，进行细粒度的处理然后再播放出来。Web Audio 也可以对 Web RTC 中的音频进行处理，Web RTC 是让两个或多个用户进行端对端的视频通信的 API。

权威参考：http://www.w3.org/TR/webaudio/

使用参考：https://developer.mozilla.org/zh-CN/docs/Web/API/Web_Audio_API

一个简单而典型的 Web Audio 流程如下：

第 1 步，创建音频上下文。

第 2 步，在音频上下文里创建源 — 例如 <audio>, 振荡器, 流。

第 3 步，创建效果节点，例如混响、双二阶滤波器、平移、压缩。

第 4 步，为音频选择一个目的地，例如你的系统扬声器。

第 5 步，连接源到效果器，对目的地进行效果输出。

Web Audio API 所涉及内容较多，本节将不会深入探讨，感兴趣的读者可以参考上面提供的网址深入学习。下面通过一个简单的示例介绍 Web Audio API 的基本用法。

【示例】下面示例利用 Web Audio API 在页面中设计一个按钮，当鼠标经过时，在非 IE 浏览器下，就会播放出类似电子琴琴键按下的声音，而且每次鼠标经过的时候，音调都会发生有规律的变化，如图 5.9 所示。



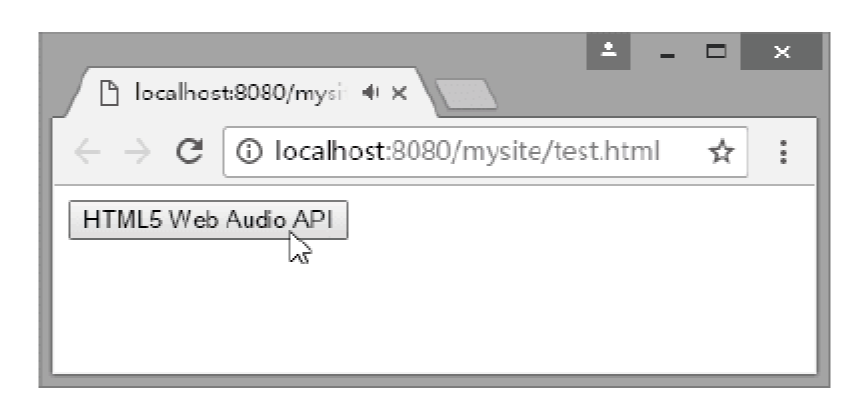


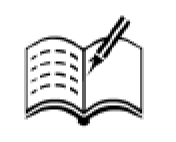
图 5.9　动态生成声音

具体代码解析请扫码学习。





视频讲解



Note

5.2.7　访问多媒体属性、方法和事件

在第 5.1.3、5.1.4、5.1.5 三节中，我们分别介绍了 HTML5 多媒体 API 中各种属性、方法和事件，本节将以示例的形式演示如何在一个视频中实现对这些信息进行访问和操控，示例效果如图 5.10 所示。

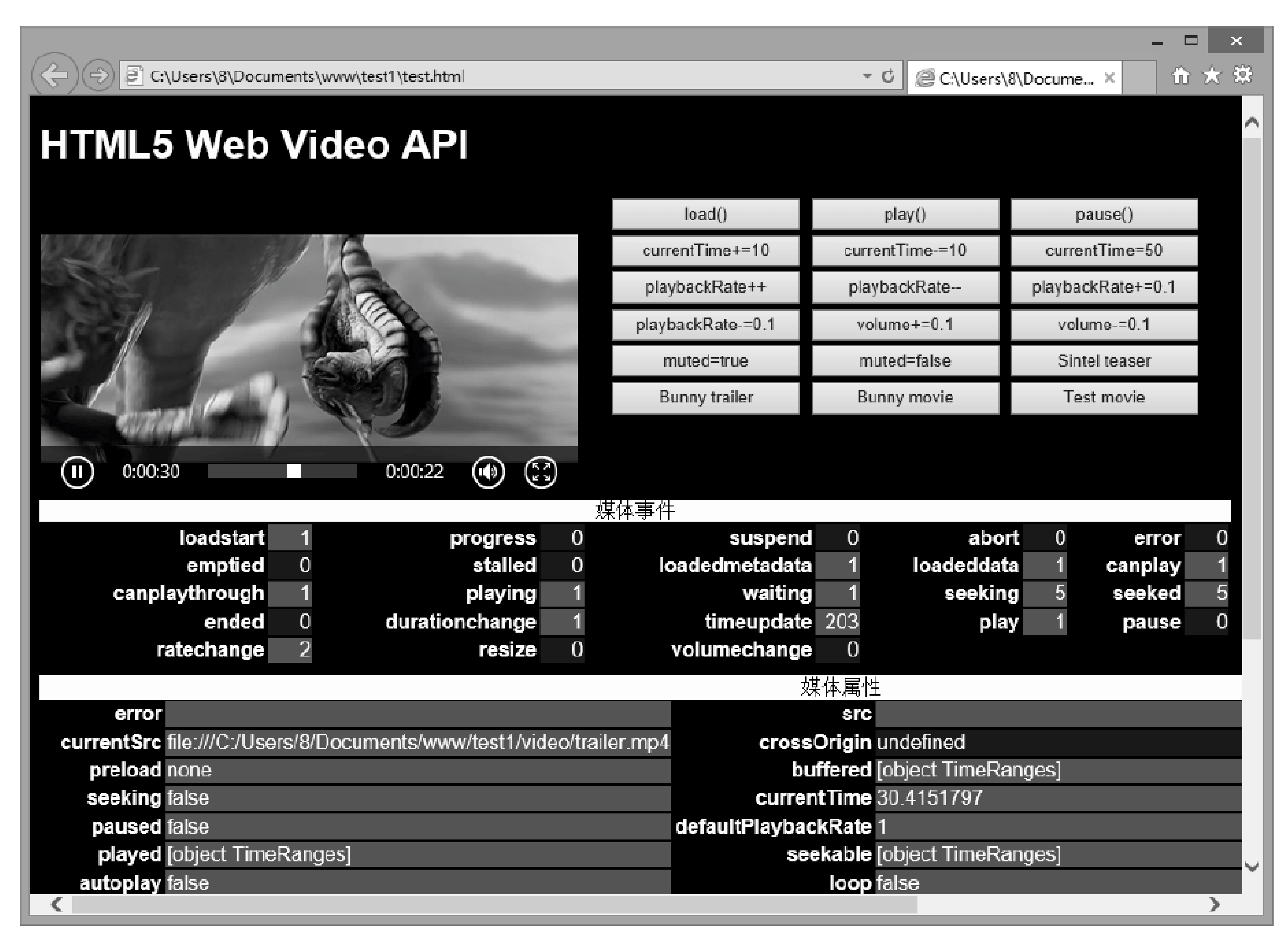


图 5.10　HTML5 多媒体 API 接口访问

具体操作步骤请扫码学习。



线上阅读

5.3　在线练习

使用多媒体丰富网站的效果，丰富网站的内容，突出网站的重点。



在线练习

第6章

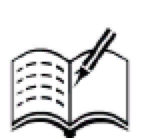
数据存储

在HTML4中，客户端处理网页数据主要通过cookie来实现，但cookie存在很多缺陷，如不安全、容量有限等。HTML5新增Web Database API，用来替代cookie解决方案，对于简单的key/value（键值对）信息，使用Web Storage存储会非常方便。另外，部分现代浏览器还支持不同类型的本地数据库，使用客户端数据库可以减轻服务器端的压力，提升 Web应用的访问速度。

【学习重点】

- 使用Web Storage。
- 使用Web SQL数据库。
- 使用indexedDB数据库。





6.1　Web Storage

HTML5 的 Web Storage API 提供了两种客户端数据存储的方法：localStorage 和 sessionStorage。两者之间的重要区别如下，具体用法基本相同。

☑ localStorage：用于持久化的本地存储，除非主动删除，否则数据永远不会过期。

☑ sessionStorage：用于存储本地会话（session）数据，这些数据只有在同一个会话周期内才能访问，当会话结束后数据也随之销毁，如关闭网页，切换选项卡视图等。因此 sessionStorage 是一种短期本地存储方式。

Web Storage 的优势：

☑ 存储空间比 cookie 大很多。

☑ 存储内容不会反馈给服务器，而 cookie 信息会随着请求一并发送给服务器。

☑ Web Storage 提供了一套丰富的接口，使得数据操作更为简便。

☑ 独立的存储空间，每个域（包括子域）有独立的存储空间，各个存储空间是完全独立的，因此不会造成数据混乱。

Web Storage 的缺陷：

☑ 浏览器不会检查脚本所在的域与当前域是否相同。例如，如果在域 B 中嵌入域 A 的脚本文件，那么域 A 的脚本文件可以访问域 B 中的数据。不过这个漏洞很容易修补，就看浏览器厂商的态度了。

☑ 存储数据未加密，且永远保存，容易泄露。

在 HTML5 众多 API 中，Web Storage 的浏览器支持是非常好的，目前主流浏览器都支持 Web Storage，如 IE 8+、Firefox 3+、Opera 10.5+、Chrome 3.0+和 Safari 4.0+。

6.1.1　使用 Web Storage



视 频 讲 解

localStorage 和 sessionStorage 对象拥有相同的属性和方法，操作方法也都相同。

1．存储

使用 setItem()方法可以存储值，用法如下：

```
setItem( key, value)
```

参数 key 表示键名，value 表示值，都以字符串形式进行传递。例如：

```
sessionStorage.setItem("key", "value");
localStorage.setItem("site", "mysite.cn");
```

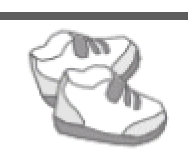
2．访问

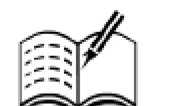
使用 getItem()方法可以读取指定键名的值，用法如下：

```
getItem(key)
```

参数 key 表示键名，字符串类型。该方法将获取指定 key 本地存储的值。例如：

```
var value = sessionStorage.getItem("key");
var site = localStorage.getItem("site");
```




Note

3．删除

使用 removeItem()方法可以删除指定键名本地存储的值。用法如下：

```
removeItem(key)
```

参数 key 表示键名，字符串类型。该方法将删除指定 key 本地存储的值。例如：

```
sessionStorage.removeItem("key");
localStorage.removeItem("site");
```

4．清空

使用 clear()方法可以清空所有本地存储的键值对。用法如下：

```
clear()
```

例如，调用 clear()方法可以直接清理本地存储的数据。

```
sessionStorage.clear();
localStorage.clear();
```


提示：Web Storage 也支持使用点语法，或者使用字符串数组[]的方式来处理本地数据。例如：

```
var storage = window.localStorage;          //获取本地 localStorage 对象
//存储值
storage.key = "hello";
storage["key"] = "world";
//访问值
console.log(storage.key);
console.log(storage["key"]);
```

5．遍历

Web Storage 定义 key()方法和 length 属性，使用它们可以对存储数据进行遍历操作。

【示例 1】下面示例获取本地 localStorage，然后使用 for 语句访问本地存储的所有数据，并输出到调试台显示。

```
var storage = window.localStorage;
for (var i=0, len = storage.length; i < len; i++){
    var key = storage.key(i);
    var value = storage.getItem(key);
    console.log(key + "=" + value);
}
```

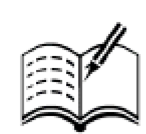
6．监测事件

Web Storage 定义 storage 事件，当键值改变或者调用 clear()方法的时候，将触发 storage 事件。

【示例 2】下面示例使用 storage 事件监测本地存储，当发生值变动时，即时进行提示。

```
if(window.addEventListener){
    window.addEventListener("storage",handle  storage,false);
 }else if(window.attachEvent){
    window.attachEvent("onstorage",handle_storage);
 }
```







```
    function handle  storage(e) {
        var logged = "key:" + e.key + ", newValue:" + e.newValue + ", oldValue:" + e.oldValue + ", url:" + e.url + ", storageArea:" + e.storageArea;
        alert(logged);
    }
```

storage 事件对象包含属性说明如表 6.1 所示。

表 6.1　storage 事件对象属性

属　　性	类　　型	说　　明
key	String	键的名称
oldValue	Any	以前的值（被覆盖的值），如果是新添加的项目，则为 null
newValue	Any	新的值，如果是新添加的项目，则为 null
url/uri	String	引发更改的方法所在页面地址

6.1.2　案例：设计登录页



视 频 讲 解

本例演示如何使用 localStorage 对象保存用户登录信息，运行结果如图 6.1 所示。

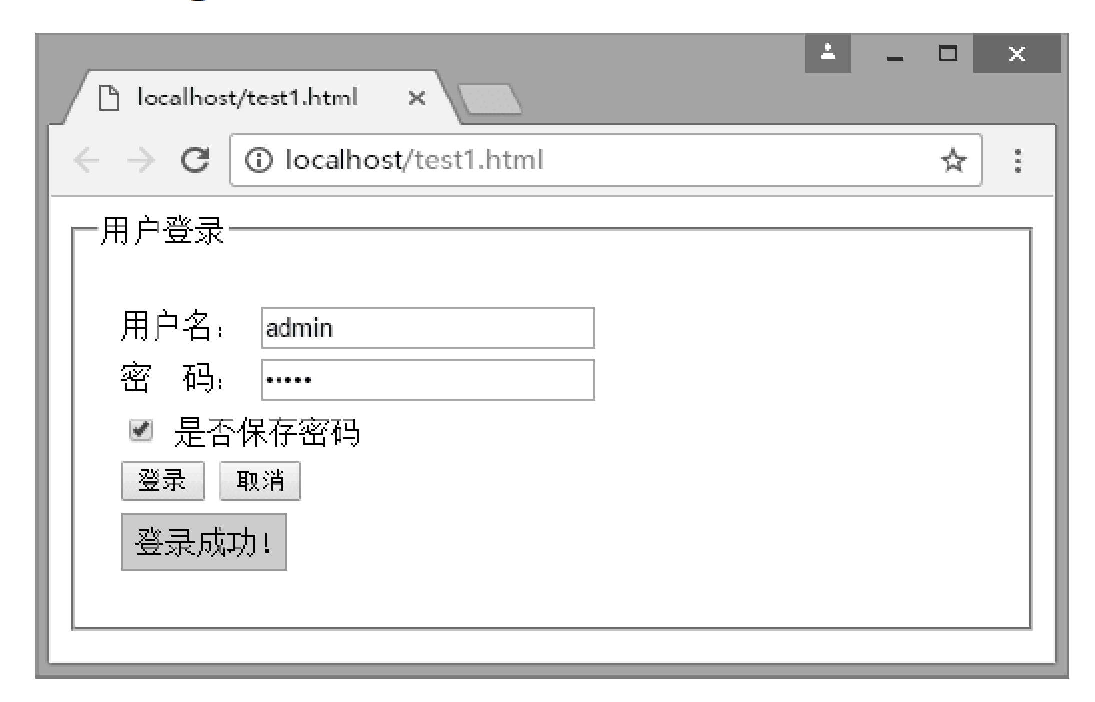


图 6.1　保存用户登录信息



示 例 效 果

当用户在文本框中输入用户名与密码，单击“登录“按钮后，浏览器将调用 localStorage 对象保存登录用户名。如果选中“是否保存密码”复选框，会同时保存密码，否则，将清空可能保存的密码。当重新打开该页面时，经过保存的用户名和密码数据将分别显示在文本框中，避免用户重复登录。

示例代码如下所示：

```
    <style type="text/css">
    ul { list-style-type: none; padding:3px 6px; }
    ul li { margin:6px; }
    .li_title { margin-top:12px;}
    .status { border: 1px solid #999999; background: #CCCCCC; padding:6px; }
    </style>
    <script type="text/javascript">
    function $(id) { return document.getElementById(id);}
    function pageload(){     //页面加载时调用的函数
        var strName=localStorage.getItem("keyName");
        var strPass=localStorage.getItem("keyPass");
```




```
        if(strName){ $("txtName").value=strName; }
        if(strPass){ $("txtPass").value=strPass; }
    }
    function btn_click(){    //单击“登录”按钮后调用的函数
        var strName=$("txtName").value;
        var strPass=$("txtPass").value;
        localStorage.setItem("keyName",strName);
        if($("chkSave").checked){ localStorage.setItem("keyPass",strPass);
        }else{localStorage.removeItem("keyPass");}
        $("spnStatus").className="status";
        $("spnStatus").innerHTML="登录成功!";
    }
    </script>

    <body onLoad="pageload();">
    <form id="frmLogin" action="#">
        <fieldset>
            <legend>用户登录</legend>
            <ul>
                <li>用户名：<input id="txtName" class="inputtxt" type="text"></li>
                <li>密 码：<input id="txtPass" class="inputtxt" type="password"></li>
                <li><input id="chkSave" type="checkbox">是否保存密码</li>
                <li><input name="btn" class="inputbtn" value="登录" type="button" onClick="btn_click();">
<input name="rst" class="inputbtn" type="reset" value="取消"> </li>
                <li class="li_title"><span id="spnStatus"></span></li>
            </ul>
        </fieldset>
    </form>
    </body>
```



视频讲解

6.1.3 案例：流量统计

本例通过 sessionStorage 和 localStorage 对页面的访问进行计数。当在文本框内输入数据后，分别单击“session 保存”按钮和“local 保存”按钮对数据进行保存，还可以通过单击“session 读取”按钮和“local 读取”按钮对数据进行读取。在 Chrome 浏览器中运行本实例的结果如图 6.2 所示。



示例效果

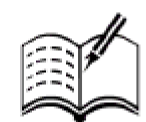


图 6.2 Web 应用计数器



示例代码如下所示：

```
<h1>计数器</h1>
<p class="msg" id="msg_1"> </p>
<p class="form_item">
    <label for="">Storage：</label>
    <input type="text" name="text-1" value="" id="text-1"/>
</p>
<p class="form_item">
    <input type="button" name="btn-1" value="session 保存" id="btn-1"/>
    <input type="button" name="btn-2" value="session 读取" id="btn-2"/>
    <input type="button" name="btn-3" value="local 保存" id="btn-3"/>
    <input type="button" name="btn-4" value="local 读取" id="btn-4"/>
</p>
<p class="count_wrap">本页 session 访问次数：<span class="count" id='session_count'></span>  
本页 local 访问次数：<span class="count" id='local_count'></span></p>
<script>
function getE(ele){    //自定义一个 getE()函数
    return document.getElementById(ele); //返回并调用 document 对象的 getElementById 方法输出变量
}
var text_1 = getE('text-1'),//声明变量并为其赋值
    mag = getE('msg_1'),
    btn_1 = getE('btn-1'),
    btn_2 = getE('btn-2'),
    btn_3 = getE('btn-3'),
    btn_4 = getE('btn-4');
btn_1.onclick = function(){sessionStorage.setItem('msg','sessionStorage = ' + text_1.value ); }
btn_2.onclick = function(){mag.innerHTML = sessionStorage.getItem('msg'); }
btn_3.onclick = function(){localStorage.setItem('msg','localStorage = ' + text_1.value );}
btn_4.onclick = function(){mag.innerHTML = localStorage.getItem('msg');}
//记录页面次数
var local_count = localStorage.getItem('a_count')?localStorage.getItem('a_count'):0;
getE('local_count').innerHTML = local_count;
localStorage.setItem('a_count',+local_count+1);
var session_count = sessionStorage.getItem('a_count')?sessionStorage.getItem('a_count'):0;
getE('session_count').innerHTML = session_count;
sessionStorage.setItem('a_count',+session_count+1);
</script>
```



Note

6.2　Web SQL Database

HTML5 新增 Web SQL Database API，允许用户使用 SQL 访问客户端数据库。该 API 不是 HTML5 规范的组成部分，而是单独的规范，它通过一套方法操纵客户端的数据库。虽然 Web SQL Database 已经在 Safari、Chrome 和 Opera 浏览器中实现，但是 IE、Firefox 浏览器并没有实现它。

由于标准认定直接执行 SQL 语句不可取，Web SQL Database 已被新规范——索引数据库（Indexed Database）所取代。WHATWG 也停止对 Web SQL Database 的开发。索引数据库更简便，而且不依赖于特定的 SQL 数据库版本。目前浏览器正在逐步实现对索引数据库的支持。





视 频 讲 解

Note

6.2.1 使用 Web SQL Database

HTML5 数据库 API 以一个独立规范形式出现，它包含三个核心方法：

☑ openDatabase：使用现有数据库或创建新数据库的方式创建数据库对象。

☑ transaction：允许我们根据情况控制事务提交或回滚。

☑ executeSql：用于执行真实的 SQL 查询。

使用 JavaScript 脚本编写 SQLLite 数据库有两个必要的步骤。

☑ 创建访问数据库的对象。

☑ 使用事务处理。

1．创建或打开数据库

首先，必须要使用 openDatabase 方法来创建一个访问数据库的对象。具体用法如下所示。

```
Database openDatabase(in DOMString name, in DOMString version, in DOMString displayName, in unsigned long estimatedSize, in optional DatabaseCallback creationCallback)
```

openDatabase 方法可以打开已经存在的数据库，如果不存在则创建。openDatabase 中五个参数分别表示：数据库名、版本号、描述、数据库大小、创建回调。创建回调没有时也可以创建数据库。

【示例 1】创建了一个数据库对象 db，名称是 Todo，版本编号为 0.1。db 还带有描述信息和大概的大小值。浏览器可使用这个描述与用户进行交流，说明数据库是用来做什么的。利用代码中提供的大小值，浏览器可以为内容留出足够的存储。如果需要，这个大小是可以改变的，所以没有必要预先假设允许用户使用多少空间。

```
db = openDatabase("ToDo", "0.1", "A list of to do items.", 200000);
```

为了检测之前创建的连接是否成功，可以检查数据库对象是否为 null：

```
if(!db)
    alert("Failed to connect to database.");
```

注意：使用中绝不可以假设该连接已经成功建立，即使过去对于某个用户它是成功的。为什么一个连接会失败，这里面存在多个原因：也许浏览器出于安全原因拒绝访问，也许设备存储有限。面对活跃而快速进化的潜在浏览器，对用户机器、软件及其能力做出假设是非常不明智的行为。如当用户使用手持设备时，他们可自由处置的数据可能只有几兆字节。

2．访问和操作数据库

实际访问数据库的时候，还需要调用 transaction 方法，用来执行事务处理。使用事务处理，可以防止在对数据库进行访问及执行有关操作的时候受到外界的打扰。因为在 Web 上，同时会有许多人都在对页面进行访问。如果在访问数据库的过程中，正在操作的数据被别的用户给修改掉，会引起很多意想不到的后果。因此，可以使用事务来达到在操作完成之前，阻止别的用户访问数据库的目的。

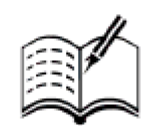
transaction 方法的使用方法如下所示。

```
db.transaction( function(tx) {})
```

transaction 方法使用一个回调函数作为参数。在这个函数中，执行访问数据库的语句。

在 transaction 的回调函数内，使用了作为参数传递给回调函数的 transaction 对象的 executeSql 方





法。executeSql 方法的完整定义如下所示。

```
transaction.executeSql(sqlquery,[],dataHandler, errorHandler);
```

该方法使用四个参数，第一个参数为需要执行的 SQL 语句。

第二个参数为 SQL 语句中所有使用到的参数的数组。在 executeSql 方法中，将 SQL 语句中所要使用到的参数先用“?”代替，然后依次将这些参数组成数组放在第二个参数中，如下所示。

```
transaction.executeSql("UPDATE people set age-? where name=?;",[age, name]);
```

第三个参数为执行 SQL 语句成功时调用的回调函数。该回调函数的传递方法如下所示。

```
function dataRandler(transaction, results){ //执行 SQL 语句成功时的处理
}
```

该回调函数使用两个参数，第一个参数为 transaction 对象，第二个参数为执行查询操作时返回的查询到的结果数据集对象。

第四个参数为执行 SQL 语句出错时调用的回调函数。该回调函数的传递方法如下所示。

```
function errorHandler(transaction,errmeg) { //执行 SQL 语句出错时的处理
}
```

该回调函数使用两个参数，第一个参数为 transaction 对象，第二个参数为执行发生错误时的错误信息文字。

【示例 2】下面将在 mydatabase 数据库中创建表 t1，并执行数据插入操作，完成插入两条记录。

```
var db = openDatabase('mydatabase', '2.0', my db', 2 * 1024);
db.transaction(function (tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
        tx.executeSql('INSERT INTO t1 (id, log) VALUES (1, "foobar")');
        tx.executeSql('INSERT INTO t1 (id, log) VALUES (2, "logmsg")');
});
```

在插入新记录时，还可以传递动态值：

```
var db = openDatabase(' mydatabase ', '2.0', 'my db', 2 * 1024);
db.transaction(function (tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
        tx.executeSql('INSERT INTO t1 (id,log) VALUES (?, ?'), [e_id, e_log];    //e_id 和 e_log 是外部变量
});
```

当执行查询操作时，从查询到的结果数据集中依次把数据取出到页面上来，最简单的方法是使用 for 语句循环。结果数据集对象有一个 rows 属性，其中保存了查询到的每条记录，记录的条数可以用 rows.length 来获取，可以用 for 循环，用 rows[index]或 rows.Item ((index])的形式来依次取出每条数据。在 JavaScript 脚本中，一般采用 rows[index]的形式。另外，在 Chrome 浏览器中不支持 rows.Item ([index]) 的形式。

【示例 3】如果要读取已经存在的记录，使用一个回调函数来捕获结果，并通过 for 语句循环显示每条记录。

```
var db = openDatabase(mydatabase, '2.0', 'my db', 2*1024);
db.transaction(function (tx) {
      tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
      tx.executeSql('INSERT INTO t1 (id, log) VALUES (1, "foobar")');
```



Note

```
        tx.executeSql('INSERT INTO t1 (id, log) VALUES (2, "logmsg")');
    });
    db.transaction(function (tx) {
        tx.executeSql('SELECT * FROM t1', [], function (tx, results) {
            var len = results.rows.length, i;
            msg = "<p>Found rows: " + len + "</p>";
            document.querySelector('#status').innerHTML +=    msg;
            for (i = 0; i < len; i++){
                alert(results.rows.item(i).log );
            }
        }, null);
    });
```

【示例 4】下面示例将完整地演示 Web SQL Database API 的使用，包括建立数据库、建立表格、插入数据、查询数据、将查询结果显示。在 Chrome、Safari 或 Opera 浏览器中输出结果如图 6.3 所示。



图 6.3　创建本地数据库

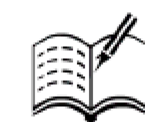
示例代码如下所示：

```
    <script type="text/javascript">
    var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
    var msg;
    db.transaction(function(tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
        tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
        tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
        msg = '<p>完成消息创建和插入行操作。</p>';
        document.querySelector('#status').innerHTML = msg;
    });
    db.transaction(function(tx) {
        tx.executeSql('SELECT * FROM LOGS', [], function(tx, results) {
            var len = results.rows.length, i;
            msg = "<p>查询行数: " + len + "</p>";
            document.querySelector('#status').innerHTML += msg;
            for( i = 0; i < len; i++) {
                msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
                document.querySelector('#status').innerHTML += msg;
            }
        }, null);
    });
    </script>
```



```
<div id="status" name="status"></div>
```

其中第二行的“var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);”建立一个名称为 mydb 的数据库，它的版本为 1.0，描述信息为 Test DB，大小为 2MB。可以看到此时有数据库建立，但并无表格建立，如图 6.4 所示。



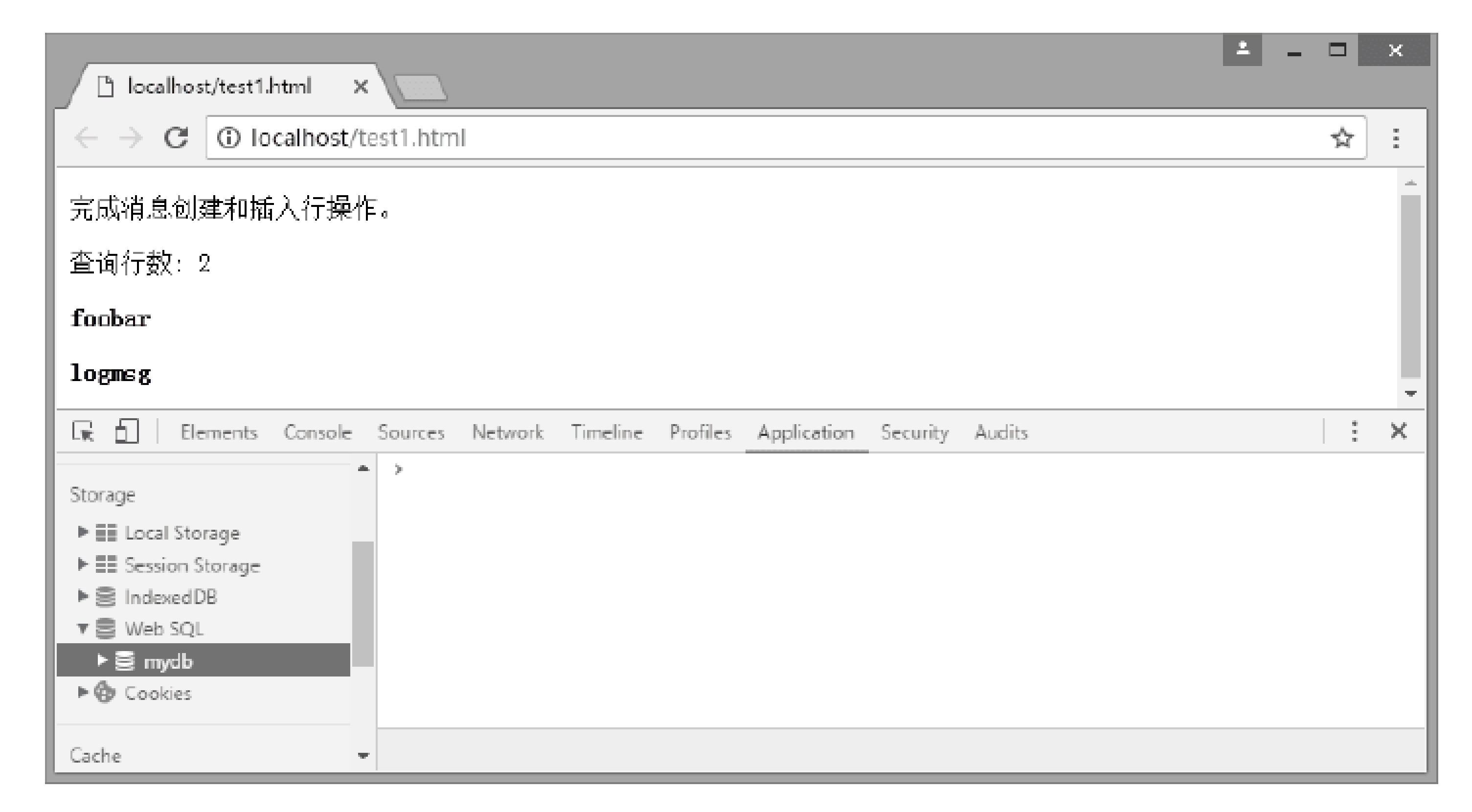


图 6.4　创建数据库 mydb

openDatabase 方法打开一个已经存在的数据库，如果数据库不存在则创建数据库，创建数据库包括数据库名、版本号、描述、数据库大小、创建回调函数。最后一个参数创建回调函数，在创建数据库的时候调用，但即使没有这个参数，一样可以运行时创建数据库。

第四行到第十行代码：

```
db.transaction(function(tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
    tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
    tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
    msg = '<p>完成消息创建和插入行操作。</p>';
    document.querySelector('#status').innerHTML = msg;
});
```

通过第五行语句可以在 mydb 数据库中建立一个 LOGS 表格。在这里只执行创建表格语句，而不执行后面两个插入操作时，在 Chrome 浏览器中可以看到在数据库 mydb 中有表格 LOGS 建立，但表格 LOGS 为空。

第六、七两行执行插入操作，在插入新记录时，还可以传递动态值：

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
    tx.executeSql('INSERT INTO LOGS (id,log) VALUES (?, ?'), [e_id, e_log];
});
```

这里的 e_id 和 e_log 为外部变量，executeSql 在数组参数中将每个变量映射到“?”。在插入操作执行后，可以在 Chrome 浏览器中看到数据库的状态，可以看到插入的数据，此时并未执行查询语句，页面中并没有出现查询结果，如图 6.5 所示。




Note

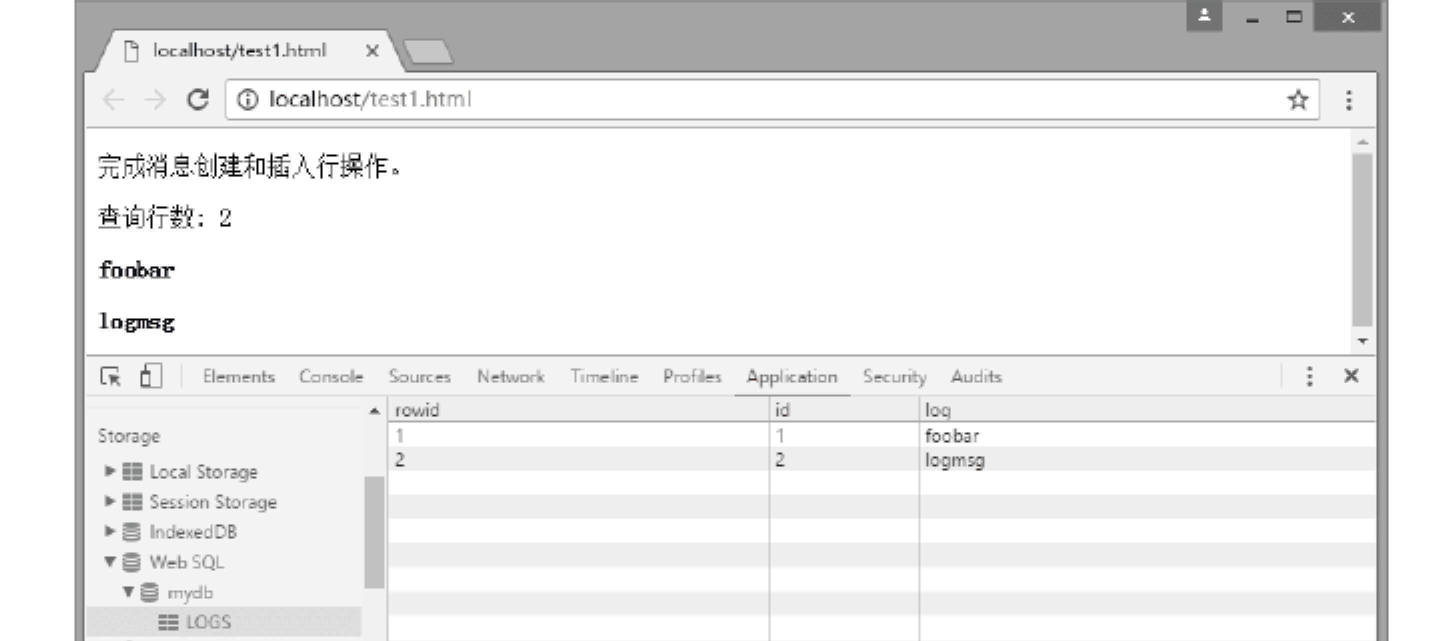


图 6.5　创建数据表并插入数据

如果要读取已经存在的记录，使用一个回调函数捕获结果，如上面的第十一到第二十一行代码：

```
db.transaction(function(tx) {
    tx.executeSql('SELECT * FROM LOGS', [], function(tx, results) {
        var len = results.rows.length, i;
        msg = "<p>查询行数: " + len + "</p>";
        document.querySelector('#status').innerHTML += msg;
        for( i = 0; i < len; i++) {
    msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
    document.querySelector('#status').innerHTML += msg;
        }
    }, null);
});
```

执行查询之后，将信息输出到页面中，可以看到页面中显示查询结果。

注意：如果不需要，不要使用 Web SQL Database，因为它会让代码更加复杂（匿名内部类的内部函数、回调函数等）。在大多数情况下，本地存储或会话存储就能够完成相应的任务，尤其是能够保持对象状态持久化的情况。通过这些 HTML5 Web SQL Database API 接口，可以获得更多功能，相信以后会出现一些非常优秀的、建立在这些 API 之上的应用程序。



视频讲解

6.2.2　案例：设计登录页

本例设计一个用户登录页面，演示如何对本地数据库进行具体操作，运行结果如图 6.6 所示。

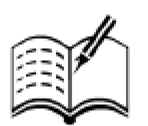




示例效果

图 6.6　用户登录





在浏览器中访问页面，然后在表单中输入用户名和密码，单击“登录”按钮，登录成功后，用户名、密码以及登录时间将显示在页面上。单击“注销”按钮，将清除已经登录的用户名、密码以及登录时间。

示例代码如下所示：

```
<style type="text/css">
body { font-size: 14px; width: 80%; margin: 6px auto; }
input[type="text"], input[type="password"]{ width: 180px; height: 24px; line-height: 24px; }
input[type="submit"], input[type="button"] { width: 80px; height: 24px; line-height: 24px; border: 1px solid
#ff6600; border-radius: 4px; background: #ff6600; outline: none; color: #fff; cursor: pointer; margin-top: 6px; }
p{margin:6px;}
</style>

<form action="#" method="get" accept-charset="utf-8">
    <h1>用户登录</h1>
    <p>用户名：<input type="text" name="" value="" id="name" required /></p>
    <p>密 码：<input type="password" name="" value="" id="msg" required /></p>
    <p><input type="submit" id="save" value="登录"/>
    <input type="submit" id="clear" value="注销"/></p>
</form>
<script>
var datalist = getE('datalist');
if(!datalist){
    datalist = document.createElement('dl');
    datalist.className = 'datalist';
    datalist.id = 'datalist';
    document.body.appendChild(datalist);
}
var result = getE('result');
var db = openDatabase('myData','1.0','test database',1024*1024);
showAllData();
db.transaction(function(tx){
    tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT,msg TEXT,time INTEGER)',[]);
})
getE('clear').onclick = function(){
    db.transaction(function(tx){
        tx.executeSql('DROP TABLE MsgData',[]);
    })
    showAllData()
}
getE('save').onclick = function(){
    saveData();
    return false;
}
function getE(ele){return document.getElementById(ele);}
function removeAllData(){
    for (var i = datalist.children.length-1; i >= 0; i--){
        datalist.removeChild(datalist.children[i]);
    }
}
```



Note

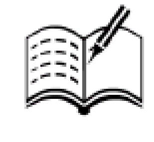
```
function showData(row){
    var dt = document.createElement('dt');
    dt.innerHTML = row.name;
    var dd = document.createElement('dd');
    dd.innerHTML = row.msg;
    var tt = document.createElement('tt');
    var t = new Date();
    t.setTime(row.time);
    tt.innerHTML =t.toLocaleDateString()+" "+ t.toLocaleTimeString();
    datalist.appendChild(dt);
    datalist.appendChild(dd);
    datalist.appendChild(tt);
}
function showAllData(){
    db.transaction(function(tx){
        tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT,msg TEXT,time INTEGER)',[]);
        tx.executeSql('SELECT * FROM MsgData',[],function(tx,result){
            removeAllData();
            for(var i=0; i < result.rows.length; i++){
                showData(result.rows.item(i));
            }
        });
    });
}
function addData(name,msg,time){
    db.transaction(function(tx){
        tx.executeSql('INSERT INTO MsgData VALUES(?,?,?)',[name,msg,time],function(tx,result){
            alert("登录成功");
        },
        function(tx,error){
            alert(error.source + ':' + error.message);
        });
    });
}
function saveData(){
    var name =getE('name').value;
    var msg = getE('msg').value;
    var time = new Date().getTime();
    addData(name,msg,time);
    showAllData();
}
</script>
```


视频讲解

6.2.3 案例：设计留言板

【示例 1】下面示例设计一个简单 Web 留言本，介绍如何使用 Web Storage 来读写留言信息。在示例页面中显示一个多行文本框，允许用户输入数据，当单击“追加”按钮时，将文本框中的数据保存到 localStorage 中，在表单下面显示一个空的 p 元素，作为数据容器动态显示用户添加的留言信息，示例效果如图 6.7 所示。









示例效果

图 6.7　使用 localStorage 存储数据的 Web 留言本

实现本例的关键是如何获取 localStorage 中的所有数据。获取 localStorage 中全部数据的时候，需要用到 localStorage 对象的两个比较重要的属性。

☑　length：所有保存在 localStorage 中的数据的条数。

☑　key(index)：将想要得到数据的索引号作为 index 参数传入，可以得到 localStorage 中与这个索引号对应的数据。

提示： Web Storage 采用键值对的形式保存数据，将文本框的内容作为值，保存时间作为键，以时间戳的形式保存，可以避免重复的键名。

示例完整代码如下所示：

```
<script type="text/javascript">
function saveStorage(id){
    var data = document.getElementById(id).value;
    var time = new Date().getTime();
    localStorage.setItem(time,data);
    alert("数据已保存。");
    loadStorage('msg');
}
function loadStorage(id){
    var result = '<table border="1">';
    for(var i = 0;i < localStorage.length;i++) {
        var key = localStorage.key(i);
        var value = localStorage.getItem(key);
        var date = new Date();
        date.setTime(key);
        var datestr = date.toGMTString();
        result += '<tr><td>' + value + '</td><td>' + datestr + '</td></tr>';
    }
    result += '</table>';
    var target = document.getElementById(id);
    target.innerHTML = result;
}
function clearStorage(){
```

```
        localStorage.clear();
        alert("全部数据被清除。");
        loadStorage('msg');
    }
    </script>

    <h1>Web 留言本</h1>
    <textarea id="memo" cols="60" rows="10"></textarea><br>
    <input type="button" value="追加" onclick="saveStorage('memo');">
    <input type="button" value="初始化" onclick="clearStorage('msg');"><hr>
    <p id="msg"></p>
```

在该页面中，除了输入数据用的文本框与显示数据用的 p 元素之外，还放置了"追加"按钮和"初始化"按钮，单击"追加"按钮来保存数据，单击"初始化"按钮来消除全部数据。

在 JavaScript 脚本部分包含三个函数：saveStorage()、loadStorage()、clearStorage()，简单说明如下。

☑ saveStorage()函数：这个函数比较简单，使用 new Date().getTime()语句得到了当前的日期和时间，然后调用 localStorage.setItem()方法，将得到的时间作为键值，并将文本框中的数据作为键名进行保存。保存完毕后，重新调用脚本中的 loadStorage()函数在页面上重新显示保存后的数据。

☑ loalStorage()函数：取得保存后的所有数据，然后以表格的形式进行显示。取得全部数据的时候，需要用到 localStorage 两个比较重要的属性。

  ➢ localStorage.length 返回所有保存在 localStorage 中的数据的条数。

  ➢ localStorage.key(index)将想要得到数据的索引号作为 index 参数传入，可以得到 localStorage 中与这个索引号对应的数据。如想得到第 6 条数据，传入的 index 为 5（index 是从 0 开始计算的）。

  先用 localStorage.length 属性获取保存数据的条数，然后做一个循环，在循环内用一个变量，从 0 开始将该变量作为 index 参数传入 localStorage.key(index)属性，每次循环时该变量加 1，通过这种方法取得保存在 localStorage 中的所有数据。

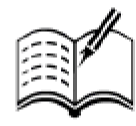
☑ clearStorage()函数：将 localStorage 中保存的数据全部清除，在这个函数中只有一句语句"localStorage.clear();"调用 localStorage 的 clear()方法时，所有保存在 localStorage 中的数据会全部被清除。

【示例 2】下面示例借助 JSON 格式数据，协助 Web Storage 实现保存二维表格式数据，本示例演示效果如图 6.8 所示。

示例代码如下所示：

```
<script type="text/javascript">
function saveStorage(){
    var data = new Object;
    data.name = document.getElementById('name').value;
    data.email = document.getElementById('email').value;
    data.tel = document.getElementById('tel').value;
    data.memo = document.getElementById('memo').value;
    var str = JSON.stringify(data);
    localStorage.setItem(data.name,str);
    alert("数据已保存。");
}
```





```
function findStorage(id){
    var find = document.getElementById('find').value;
    var str = localStorage.getItem(find);
    var data =   JSON.parse(str);
    var result = "姓名: " + data.name + '<br>';
    result += "EMAIL: " + data.email + '<br>';
    result += "电话号码: " + data.tel    + '<br>';
    result += "备注: " + data.memo + '<br>';
    var target = document.getElementById(id);
    target.innerHTML = result;
}
</script>

<h1>使用 Web Storage 模拟数据库</h1>
<table>
    <tr><td>姓名:</td><td><input type="text" id="name"></td></tr>
    <tr><td>EMAIL:</td><td><input type="text" id="email"></td></tr>
    <tr><td>电话号码:</td><td><input type="text" id="tel"></td></tr>
    <tr><td>备注:</td><td><input type="text" id="memo"></td></tr>
    <tr>
        <td></td>
        <td><input type="button" value="保存" onclick="saveStorage();"></td>
    </tr>
</table><hr>
<p>检索:<input type="text" id="find">
        <input type="button" value="检索" onclick="findStorage('msg');">
</p>
<p id="msg"></p>
```





示 例 效 果

图 6.8　使用 Web Storage 模拟数据库

本例关键技术点：使用 JSON 对象的 stringify()方法和 parse()方法把 JSON 对象转换为字符串表示，或者把数据字符串转换为 JSON 对象。

提示： 支持 JSON 对象的浏览器包括 IE 8+、Firefox 3.6+、Chrome +、Safari 5+、Opera 10+版本的浏览器。



在 JavaScript 脚本部分包含两个函数，分别是保存数据用的 saveStorage()函数与检索数据用的 findStorage()函数。



saveStorage()函数中的流程如下：

第 1 步，从各输入文本框中获取数据。

第 2 步，创建对象，将获取的数据作为对象的属性进行保存。

第 3 步，将对象转换成 JSON 格式的文本数据。

第 4 步，将文本数据保存在 localStorage 中。

为了将数据保存在一个对象中，使用 new Object 语句创建了一个对象，将各种数据保存在该对象的各个属性中，然后，为了将对象转换成 JSON 格式的文本数据，使用了 JSON 对象 stringify()方法，该方法的使用方法如下所示。

```
var str = JSON. stringify(data);
```

该方法接收一个参数 data，它表示要转换成 JSON 格式文本数据的对象，这个方法的作用是将对象转换成 JSON 格式的文本数据，并将其返回。

findStorage()函数中的流程如下：

第 1 步，在 localStorage 中将检索用的姓名作为键值，获取对应的数据。

第 2 步，将获取的数据转换成 JSON 对象。

第 3 步，取得 JSON 对象的各属性值，创建要输出的内容。

第 4 步，将要输出的内容在页面上输出。

该函数的关键是使用 JSON 对象的 parse 方法，将从 localStorage 中获取的数据转换成 JSON 对象。该方法的使用方法如下所示。

```
var data = JSON.parse(str);
```

该方法接收一个参数 str，它表示从 localStorage 中取得的数据，该方法的作用是将传入的数据转换成 JSON 对象，并且将该对象返回。

【示例 3】下面示例利用 Web SQL 数据库实现留言本的功能。设计页面中包含一个输入姓名用的文本框，一个输入留言用的文本框，以及一个保存数据时用的按钮。在按钮下面放置了一个表格，保存数据后从数据库中重新取得所有数据，然后把数据显示在这个表格中。

单击“保存”按钮时，调用 saveData()函数，保存数据时的处理都被写在了这个函数里。打开页面时将调用 init()函数，将数据库中全部已保存的留言信息显示在表格中，示例演示效果如图 6.9 所示。

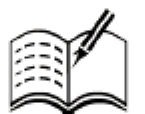


示 例 效 果



图 6.9　使用 Web SQL 设计 Web 留言本





示例代码如下所示：

```
<script type="text/javascript">
var datatable = null;
var db = openDatabase('MyData', '', 'My Database', 102400);
function init(){
    datatable = document.getElementById("datatable");
    showAllData();
}
function removeAllData(){
    for (var i =datatable.childNodes.length-1; i>=0; i--){
        datatable.removeChild(datatable.childNodes[i]);
    }
    var tr = document.createElement('tr');
    var th1 = document.createElement('th');
    var th2 = document.createElement('th');
    var th3 = document.createElement('th');
    th1.innerHTML = '姓名';
    th2.innerHTML = '留言';
    th3.innerHTML = '时间';
    tr.appendChild(th1);
    tr.appendChild(th2);
    tr.appendChild(th3);
    datatable.appendChild(tr);
}
function showData(row) {
    var tr = document.createElement('tr');
    var td1 = document.createElement('td');
    td1.innerHTML = row.name;
    var td2 = document.createElement('td');
    td2.innerHTML = row.message;
    var td3 = document.createElement('td');
    var t = new Date();
    t.setTime(row.time);
    td3.innerHTML=t.toLocaleDateString()+" "+t.toLocaleTimeString();
    tr.appendChild(td1);
    tr.appendChild(td2);
    tr.appendChild(td3);
    datatable.appendChild(tr);
}
function showAllData(){
    db.transaction(function(tx) {
        tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT, message TEXT, time 
INTEGER)',[]);
        tx.executeSql('SELECT * FROM MsgData', [], function(tx, rs) {
            removeAllData();
            for(var i = 0; i < rs.rows.length; i++){
                showData(rs.rows.item(i));
            }
        });
    });
```





Note

```
}
function addData(name, message, time) {
      db.transaction(function(tx) {
            tx.executeSql('INSERT INTO MsgData VALUES(?, ?, ?)',[name, message, time],function(tx, rs)
            {
                  alert("成功保存数据!");
            },
            function(tx, error) {
                  alert(error.source + "::" + error.message);
            });
      });
}
function saveData(){
      var name = document.getElementById('name').value;
      var memo = document.getElementById('memo').value;
      var time = new Date().getTime();
      addData(name,memo,time);
      showAllData();
}
</script>

<body onload="init();">
<h1>使用 Web SQL 设计 Web 留言本</h1>
<table>
      <tr><td>姓名:</td><td><input type="text" id="name"></td></tr>
      <tr><td>留言:</td><td><input type="text" id="memo"></td></tr>
      <tr>
<td></td>
<td><input type="button" value="保存" onclick="saveData();"></td></tr>
</table><hr>
<table id="datatable" border="1"></table>
<p id="msg"></p>
```

下面重点分析 JavaScript 脚本部分。

☑ 打开数据库

打开数据库的代码如下所示。

```
var datatable = null;
var db = openDatabase('MyData', '', 'My Database', 102400);
```

在 JavaScript 脚本一开始，使用了一个变量 datatable。用这个变量来代表页面中的 table 元素。db 变量代表使用 openDatabase()方法创建的数据库访问对象。在示例中创建了 MyData 数据库并对其进行访问。

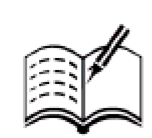
☑ 初始化

编写 init()函数，该函数在页面打开时调用。为了在打开页面时就往页面表格中装入数据，在该函数中首先设定变量 datatable 为页面中的表格，然后调用脚本中另一个函数 showAllData()来显示数据。

☑ 清除表格中当前显示的数据

removeAllData 函数是在 showAllData()函数中被调用的一个必不可少的函数，它的作用是将页面中 table 元素下的子元素全部清除，只留下一个空表格框架，然后输入表头。这样在页面表格中当前显示的数据就全部被清除了，以便重新读取数据并装入表格。





☑ 显示数据

showData()函数使用一个 row 参数，该参数表示从数据库中读取到的一行数据。该函数在页面表格中使用 tr 元素添加一行，并使用 td 元素添加各列，然后将传入的这行数据分别填入在表格中添加的这一行对应的各列中。

☑ 显示全部数据

showAllData()函数使用 transaction()方法，在该方法的回调函数中执行 executeSgl()方法获取全部数据。获取到数据之后，首先调用 removeAllData()函数初始化页面表格，将该表格中当前显示的数据全部清除，然后在循环中调用 showData()函数，将获取到的每一条数据作为参数传入，在页面上的表格中逐条显示获取到的每条数据。

☑ 追加数据

addData()函数在 saveData()函数中被调用。在 addData()函数中，使用 transaction()方法，在该方法的回调函数中执行 executeSgl 方法，将作为参数传入进来的数据保存在数据库中。

☑ 保存数据

saveData()函数先调用 addData()函数追加数据，再调用 showAllData()函数重新显示表格中的全部数据。

6.3 indexedDB

与 Web SQL Database 不同，indexedDB 是对象型数据库。与 Web Storage 和文件系统 API 一样，indexedDB 数据库的作用域也被限制在包含它的文档源中：两个同源的 Web 页面可以互相访问对方的数据，但是非同源的页面则不行。

在 IndexedDB API 中，一个数据库其实就是一个命名的对象仓库的集合。每个对象都必须有一个键（key），通过该键实现在存储区内进行该对象的存储和获取。键必须是唯一的，同一个存储区中的两个对象不能有同样的键，并且它们必须按照自然顺序存储，以便查询。

目前，Chrome 11+、Firefox 4+、Opera 18+、Safari 8+以及 IE10+版本的浏览器都支持 IndexedDB API。

6.3.1 建立连接



IndexedDB API 操作步骤如下所示：

第 1 步，通过指定名字打开 indexedDB 数据库。

第 2 步，创建一个事务对象，使用该对象在数据库中通过指定名字查询对象存储区。

第 3 步，调用对象存储区的 get()方法来查询对象，或者调用 put()方法来存储新的对象。

如果要避免覆盖已存在对象的情况，可以调用 add()方法。

如果想要查询表示键值范围的对象，创建一个 IDBRange 对象，并将其传递给对象仓库的 openCursor()方法。

如果想要使用次键进行查询的话，查询对象仓库中的命名索引，然后调用索引对象上的 get()方法或者 openCursor()方法。

使用 indexedDB 数据库时，首先需要预定义 indexedDB 数据库、该数据库所用的事务、IDBKeyRange 对象和游标对象。为了能够在各浏览器中正常运行，可以按如下代码针对各浏览器统一进行定义。




Note

```
window.indexedDB = window.indexedDB || window.webkitIndexedDB ||
                          window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction ||
                          window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange ||
                          window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
```

【示例】使用 indexedDB 数据库的时候，首先需要连接某个 indexedDB 数据库。下面示例代码演示了如何连接到 indexedDB 数据库。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
function connectDatabase(){
    var dbName = 'indexedDBTest';//数据库名
    var dbVersion =20170603; //版本号
    var idb;
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
        idb = e.target.result;
        alert('数据库连接成功');
    };
    dbConnect.onerror = function(){
        alert('数据库连接失败');
    };
}
</script>

<input type="button" value="连接数据库" onclick="connectDatabase();"/>
```

在浏览器中预览，单击“连接数据库”按钮，可以连接到 indexedDBTest 数据库，效果如图 6.10 所示。

示例效果

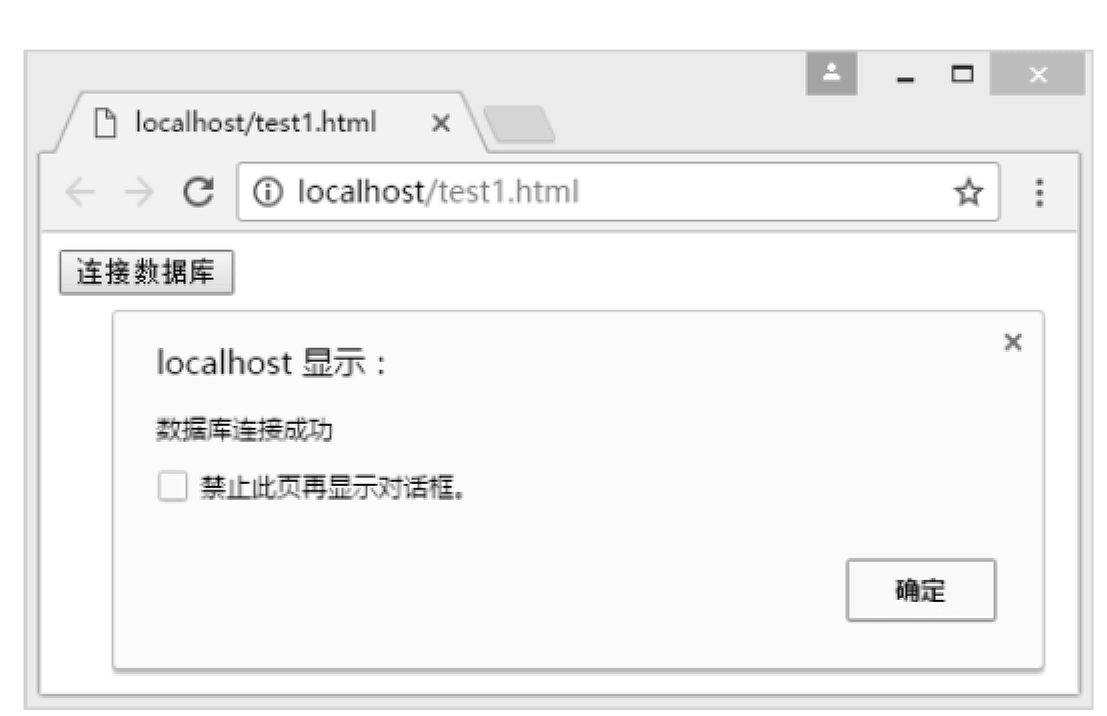
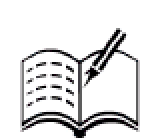


图 6.10　连接到数据库



在上面示例代码中，首先使用 indexedDB.open()方法连接数据库。该方法包含两个参数；其中第一个参数值为一个字符串，代表数据库名；第二个参数值为一个无符号长整型数值，代表数据库的版本号。indexedDB.open()方法返回一个 IDBOpenDBRequest 对象，代表一个请求连接数据库的请求对象。



然后，通过监听数据库连接的请求对象的 onsuccess 事件和 onerror 事件来定义数据库连接成功时与数据库连接失败时所需执行的事件处理函数。

在连接成功的事件处理函数中，取得事件对象的 e.target.result 属性值，该属性值为一个 IDBDatabase 对象，代表连接成功的数据库对象。

提示：在 Firefox 浏览器中访问示例页面，需要将示例页面放置在虚拟服务器运行环境。

在 IndexedDB API 中，可以通过 indexedDB 数据库对象的 close()方法关闭数据库连接，代码如下所示：

```
idb.close();
```

当数据库连接被关闭后，不能继续执行任何对该数据库进行的操作，否则浏览器均抛出异常。

6.3.2　更新版本



视频讲解

成功连接数据库之后，还不能执行任何数据操作，用户还应该创建对象仓库，以及用于检索数据的索引。这里的对象仓库相当于关系型数据库中的数据表。

在 indexedDB 数据库中，所有数据操作都必须在一个事务内部执行。事务分为三种：只读事务、读写事务和版本更新事务。

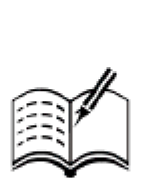
对于创建对象仓库和索引的操作，只能在版本更新事务内部进行，因为在 indexedDB API 中不允许数据库中的数据仓库在同一个版本中发生变化，所以当创建或删除数据仓库时，必须使用新的版本号来更新数据库的版本，以避免重复修改数据库结构。

对于数据库的版本更新处理，在 HTML5 中包括 2011 年 12 月之前和 2011 年 12 月之后两种不同的版本，在 Chrome 10～22 版中使用 2011 年 12 月之前的版本，在 Firefox 4+、Chrome 23+、Opera 18+、Safari 8 和 IE 10+版本的浏览器中使用 2011 年 12 月之后的版本。

【示例】下面示例只针对 2011 年 12 月之后的版本进行演示介绍。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
function VersionUpdate(){
    var dbName = 'indexedDBTest'; //数据库名
    var dbVersion = 20170603; //版本号
    var idb;
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
        idb = e.target.result;
        alert('数据库连接成功');
    };
```





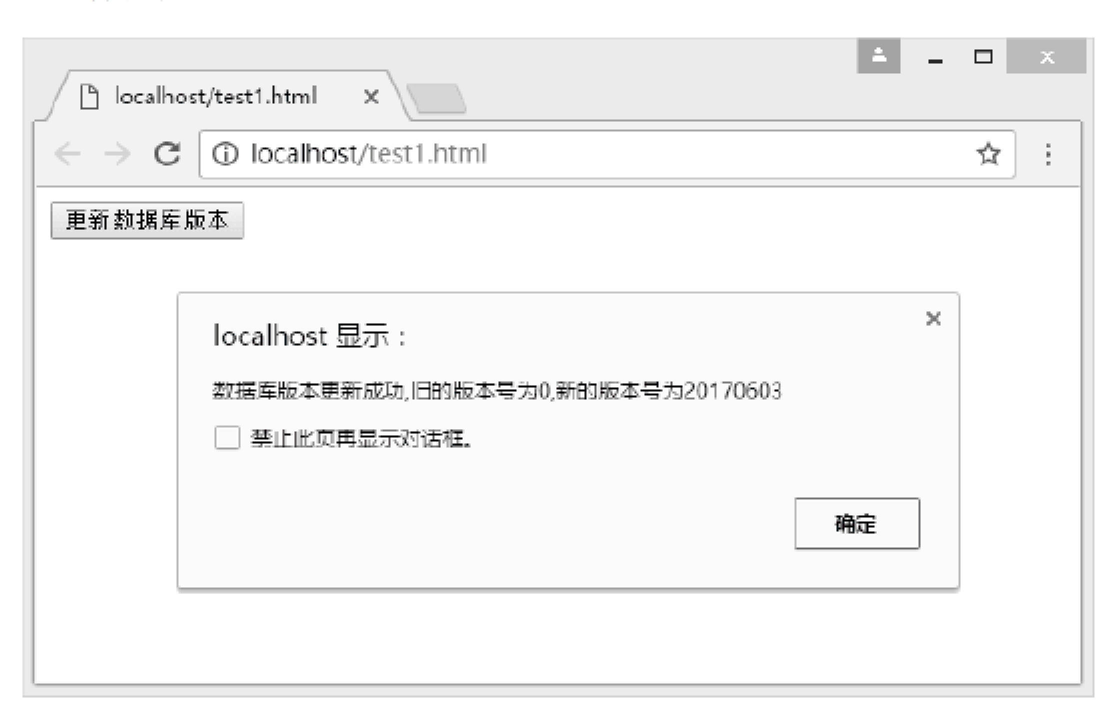
Note

```
        dbConnect.onerror = function(){
            alert('数据库连接失败');
        };
        dbConnect.onupgradeneeded = function(e){
            //数据库版本更新
            //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
            idb = e.target.result;
            /*e.target.transaction 属性值为一个 IDBTransaction 事务对象，此处代表版本更新事务*/
            var tx = e.target.transaction;
            var oldVersion = e.oldVersion; //更新前的版本号
            var newVersion = e.newVersion; //更新前的版本号
            alert('数据库版本更新成功,旧的版本号为'+oldVersion+',新的版本号为'+newVersion);
        };
    }
    </script>

    <input type="button" value="更新数据库版本" onclick="VersionUpdate();"/>
```

上面代码监听数据库连接的请求对象的 onupgradeneeded 事件，当连接数据库时发现指定的版本号大于数据库当前版本号时将触发该事件，当该事件被触发时一个数据库的版本更新事务已经被开启，同时数据库的版本号已经被自动更新完毕，并且指定在该事件触发时所执行的处理，该事件处理函数就是版本更新事务的回调函数。

在浏览器中预览页面，单击页面中的“更新数据库版本”按钮，将弹出提示信息，提示用户数据库版本更新成功，如图 6.11 所示。





示 例 效 果

图 6.11　更新数据库版本

6.3.3　新建仓库

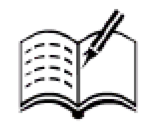


视 频 讲 解

针对 indexedDB API 中的版本更新处理，在 Chrome 10～22 版本的浏览器中使用 2011 年 12 月之前的版本，在 Chrome 23+、Opera 18+、IE 10+、Firefox 4+和 Safari 8+版本的浏览器中均使用 2011 年 12 月之后的版本，下面示例针对第二种版本介绍如何创建对象仓库。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
```





```
function CreateObjectStore(){
    var dbName = 'indexedDBTest'; //数据库名
    var dbVersion = 20170305; //版本号
    var idb;
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
        idb = e.target.result;
        alert('数据库连接成功');
    };
    dbConnect.onerror = function(){alert('数据库连接失败'); };
    dbConnect.onupgradeneeded = function(e){
        //数据库版本更新
        //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
        idb = e.target.result;
        /*e.target.transaction 属性值为一个 IDBTransaction 事务对象，此处代表版本更新事务*/
        var tx = e.target.transaction;
        var name = 'Users';
        var optionalParameters = {
            keyPath: 'userId',
            autoIncrement: false
        };
        var store = idb.createObjectStore(name,  optionalParameters);
        alert('对象仓库创建成功');
    };
}
</script>

<input type="button" value="创建对象仓库" onclick="CreateObjectStore();" />
```

上面代码监听数据库连接的请求对象的 onupgradeneeded 事件，并且指定在该事件触发时调用数据库对象的 createObjectStore()方法创建对象仓库。

createObjectStore()方法包含两个参数：第一个参数值为一个字符串，代表对象仓库名；第二个参数为可选参数 optionalParameters，参数值为一个 JavaScript 对象，该对象的 keyPath 属性值用于指定对象仓库中的每一条记录使用哪个属性值来作为该记录的主键值。

一条记录的主键为数据仓库中该记录的唯一标识符，在一个对象仓库中只能有一个主键，但是主键值可以重复，相当于关系型数据库中数据表的 id 字段为数据表的主键，多条记录的 id 字段值可以重复，除非将主键指定为唯一主键。

在 indexedDB API 中，对象仓库中的每一条记录均为具有一个或多个属性值的一个对象，而 keyPath 属性值用于指定每一条记录使用哪个属性值作为该记录的主键值。例如，在这里将数据记录的 userId 属性值作为每条记录的主键值，相当于在关系型数据库中将每条记录的 userId 字段值指定为该记录的主键值。

在这种情况下，因为主键存在于每条记录内部，所以被称为内联主键，如果在这里不指定 keyPath 属性值，或将其指定为 null，每条记录的主键将通过其他途径被另行指定，这时因为数据记录的主键存在于每条记录之外，所以被称为外部主键。

optionalParameters 对象的 autoincrement 属性值为 true，相当于在关系型数据库中将主键指定为自

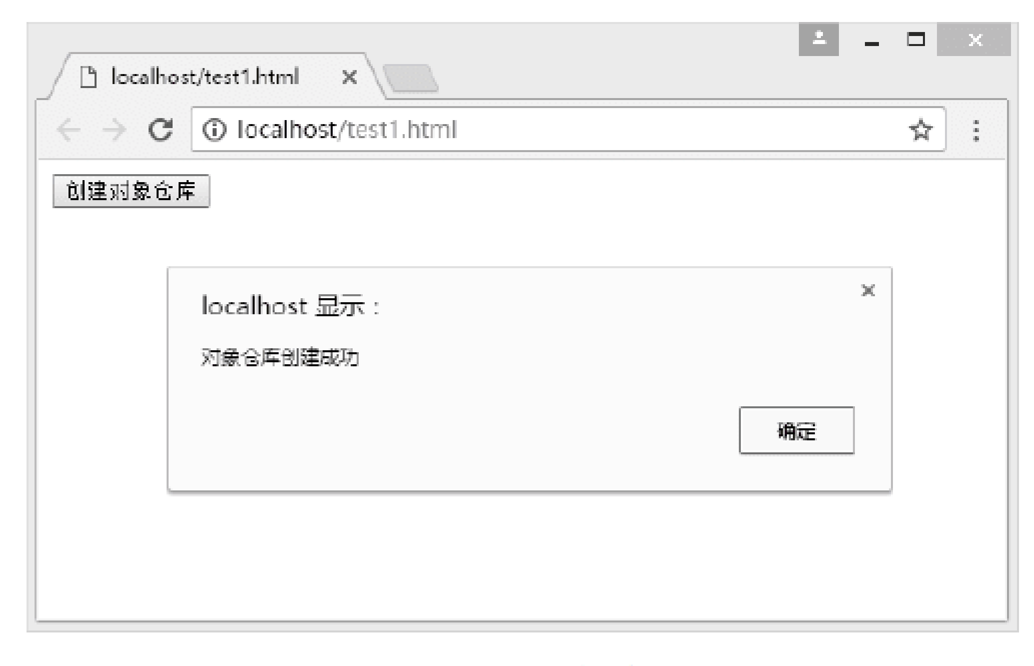


增主键，如果添加数据记录时不指定主键值，则在数据仓库内部将自动指定该主键值为既存的最大主键值+1。也可以在添加数据记录时显式地指定主键值。如果将 optionalParameters 对象的 autoIncrement 属性值指定为 false，则必须在添加数据记录时显式地指定主键值。



createObjectStore()方法返回一个 IDBObjectStore 对象，该对象代表被创建成功的对象仓库。

在 Chrome 浏览器中打开示例页面，单击页面中的“创建对象仓库”按钮，弹出提示信息，提示用户 users 对象仓库创建成功，如图 6.12 所示。





示例效果

图 6.12　创建对象仓库成功

6.3.4　新建索引

视频讲解

indexedDB 数据库中的索引类似于关系型数据库中的索引，需要通过数据记录对象的某个属性值来创建。在 indexedDB 数据库中创建索引之后，可以提高在对数据仓库中的所有数据记录进行检索时的性能。

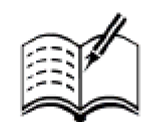
在关系型数据库中，可以针对非索引字段进行检索，而在 indexedDB 数据库中，只能针对被设为索引的属性值进行检索。

针对 indexedDB API 中的版本更新处理，分为在 Chrome 18～22 版本的浏览器中使用的 2011 年 12 月之前的版本，在 Chrome 23+、Opera 18+、IE 10+、Firefox 4+和 Safari 8+版本的浏览器中使用的 2011 年 12 月之后的版本。

【示例】下面示例只针对第二种版本进行介绍。在 indexedDB 数据库中，不能重复创建同名的对象仓库，所以在本示例中将对象仓库名修改为 newUsers，避免在运行完上节示例之后继续运行代码时浏览器抛出异常。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
function CreateIndex(){
    var dbName = 'indexedDBTest'; //数据库名
    var dbVersion = 20150306; //版本号
    var idb;
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
```





```
            idb = e.target.result;
            alert('数据库连接成功');
        };
        dbConnect.onerror = function(){
            alert('数据库连接失败');
        };
        dbConnect.onupgradeneeded = function(e){
            //数据库版本更新
            //e.target.result 为一个 IDBDatabase 对象，代表连接成功的数据库对象
            idb = e.target.result;
            /*e.target.transaction 属性值为一个 IDBTransaction 事务对象，此处代表版本更新事务*/
            var tx = e.target.transaction;
            var name = 'newUsers';
            var optionalParameters = {
                keyPath: 'userId',
                autoIncrement: false
            };
            var store = idb.createObjectStore(name,   optionalParameters);
            alert('对象仓库创建成功');
            var name =   'userNameIndex';
            var keyPath = 'userName';
            var optionalParameters = {
                unique: false,
                multiEntry: false
            };
             var idx = store.createIndex(name, keyPath, optionalParameters);
            alert('索引创建成功');
        };
    }
    </script>

    <input type="button" value="创建索引" onclick="CreateIndex();"/>
```

在数据库的版本更新事务中，在对象仓库创建成功后，调用对象仓库的 createIndex()方法创建索引。该方法包含三个参数：

第一个参数值为一个字符串，代表索引名。

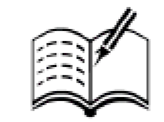
第二个参数值代表使用数据仓库中数据记录对象的哪个属性来创建索引。在本示例代码中，虽然索引名与用于创建索引的属性名不同，但是实际上此处索引名与属性名也可以相同，例如，此处可以将索引名定义为 userName。

第三个参数 optionalParameters 为可选参数，参数值为一个 JavaScript 对象，该对象的 unique 属性值的作用相当于关系型数据库中索引的 unique 属性值的作用。属性值为 true，代表同一个对象仓库中两条数据记录的索引属性值（即 userName 属性值）不能相同，否则在向数据仓库中添加第二条数据记录时将导致添加失败。

optionalParameters 对象的 multiEntry 属性值为 true，代表当数据记录的索引属性值为一个数组时，可以将数组中的每个元素添加在索引中；multiEntry 属性值为 false，代表只能将该数组整体添加在索引中。

createIndex()方法返回一个 IDBIndex 对象，代表创建索引成功。





Note

在 Chrome 浏览器中打开示例页面，单击页面中的“创建索引”按钮，弹出提示信息，提示用户索引创建成功，如图 6.13 所示。

示例效果



图 6.13　创建索引成功



视频讲解

6.3.5　使用事务

在 indexedDB API 中，所有针对数据的操作都只能在一个事务中被执行。indexedDB 提供三类事务模式，简单说明如下：

☑ readonly：只读。提供对某个对象存储的只读访问，在查询对象存储时使用。

☑ readwrite：读写。提供对某个对象存储的读取和写入访问权。

☑ versionchange：数据库版本更新。提供读取和写入访问权来修改对象存储定义，或者创建一个新的对象存储。

默认的事务模式为 readonly。用户可在任何给定时刻内打开多个并发的 readonly 事务，但只能打开一个 readwrite 事务。出于此原因，只有在数据更新时才考虑使用 readwrite 事务。单独的（表示不能打开任何其他并发事务）versionchange 事务操作一个数据库或对象存储。可以在 onupgradeneeded 事件处理函数中使用 versionchange 事务创建、修改或删除一个对象仓库，或者将一个索引添加到对象仓库。

在 indexedDB API 中，使用某个已建立连接的数据库对象的 transaction()方法可以开启事务。例如，要在 readwrite 模式下为 employees 对象仓库创建一个事务：

```
var transaction = db.transaction("employees", "readwrite");
```

transaction()方法包含两个参数：

☑ 第一个参数为由一些对象仓库名组成的一个字符串数组，用于定义事务的作用范围，即限定当事务中的数据存取操作只针对某个对象仓库进行时该事务中所运行的读写操作只能针对哪些对象仓库进行。

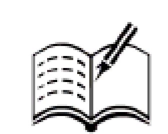
> **提示：** 如果不想限定事务只针对哪些对象仓库进行，那么可以使用数据库的 objectStoreNames 属性值来作为 transaction 方法的第一个参数值，代码如下所示：
>
> ```
> var transaction = db.transaction(idb.objectStoreNames, "readwrite");
> ```

数据仓库的 objectStoreNames 属性值为由该数据库中所有对象仓库名构成的数组，在将其作为 transaction()方法的第一个参数值时，可以针对数据库中任何一个对象仓库进行数据的存取操作。

☑ 第二个参数为可选参数，用于定义事务的读写模式，即指定事务为只读事务，还是读写事务。



transaction()方法返回一个 IDBTransaction 对象，代表被开启的事务。



注意：在将数据库的 objectStoreNames 属性值作为 transaction()方法的第一个参数值，将"readwrite"常量值作为 transaction()方法的第二个参数值，运行 transaction()方法之后，虽然在接下来的代码中可以不必再注意事务针对哪些对象仓库进行，以及事务为只读事务，还是读写事务，但是这种做法将对事务在运行时的性能产生很大的不利影响。考虑到运行时的性能，建议应该正确指定事务的作用范围，以及事务的读写模式。

提示：在 indexedDB API 中，可以同时运行多个作用范围不重叠的读写事务，如果数据库中存在 storeA 和 storeB 两个对象仓库，事务 A 的作用范围为 storeA，事务 B 的作用范围为 storeB，那么可以同时运行事务 A 和事务 B。如果将事务 A 的作用范围修改为同时包括 storeA 和 storeB 两个对象仓库，且先运行事务 A，那么事务 B 必须等到事务 A 运行结束后才能运行。即使事务 A 为只读事务，仍然可以同时运行事务 A 和事务 B。

在 indexedDB API 中，用于开启事务的 transaction()方法必须被书写到某一个函数中，而且该事务将在函数结束时被自动提交，所以不需要显式调用事务的 commit()方法来提交事务，但是可以在需要的时候显式调用事务的 abort()方法来中止事务。

可以通过监听事务对象的 oncomplete 事件（事务结束时触发）和 onabort 事件（事务中止时触发），并定义事件处理函数来定义事务结束或中止时所要执行的处理。例如：

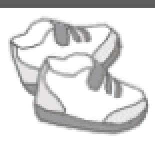
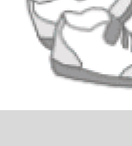
```
var transaction = db.transaction(idb.objectStoreNames, "readwrite");
transaction.oncomplete = function(event){      //事务结束时所要执行的处理
}
transaction.onabort = function(event){         //事务中止时所要执行的处理
}
//事务中的处理内容
transaction.abort();                           //中止事务
```

6.3.6　保存数据



本节介绍如何在 indexedDB 数据库的对象仓库中保存数据，示例代码如下所示。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
function SaveData(){
    var dbName = 'indexedDBTest'; //数据库名
    var dbVersion = 20170306; //版本号
    var idb;
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        idb = e.target.result; //引用 IDBDatabase 对象
        var tx = idb.transaction(['users'],"readwrite"); //开启事务
        var store = tx.objectStore('users');
```

```
            console.log(store); //-> {IDBObjectStore}
            var value = {
                userId: 1,
                userName: '张三',
                address: '北京'
            };
            var req = store.put(value);
            req.onsuccess = function(e){ alert("数据保存成功");};
            req.onerror = function(e){ alert("数据保存失败"); };
        };
        dbConnect.onerror = function(){alert('数据库连接失败');};
    }
    </script>

    <input type="button" value="保存数据" onclick="SaveData();"/>
```

【代码解析】

第 1 步，为了保存数据，首先需要连接某个 indexedDB 数据库，并且在连接成功后使用该数据库对象的 transaction()方法开启一个读写事务。

第 2 步，使用 transaction()方法返回的被开启的事务对象的 objectStore()方法获取该事务对象的作用范围中的某个对象仓库。

```
var store = tx.objectStore('users');
```

该方法包含一个参数，参数值为所需获取的对象仓库的名称。该方法返回一个 IDBObjectStore 对象，代表获取成功的对象仓库。

第 3 步，使用该对象仓库的 put()方法向数据库发出保存数据到对象仓库中的请求。

```
var value = {
    userId: 1,
    userName: '张三',
    address: '北京'
};
var req = store.put(value);
```

在上面代码中，put()方法使用一个参数，参数值为一个需要被保存到对象仓库中的对象。put()方法返回一个 IDBRequest 对象，代表一个向数据库发出的请求。

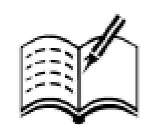
第 4 步，该请求发出后将被立即异步执行，可以通过监听请求对象的 onsuccess 事件（请求被执行成功时触发）和请求对象的 onerror 事件（请求被执行失败时触发），并指定事件处理函数来定义请求被执行成功或被执行失败时所要进行的处理。

```
req.onsuccess = function(e){
    alert("数据保存成功");
};
req.onerror = function(e){
    alert("数据保存失败");
};
```

根据对象仓库的主键是内联主键，还是外部主键，主键是否被指定为自增主键，对象仓库的 put()方法的第一个参数值的指定方法也各不相同，具体指定方法如下所示：



```
//主键为自增、内联主键时不需要指定主键值
store.put({ userName: '张三', address: '北京' });
//主键为内联、非自增主键时需要指定主键值
store.put({userId: 1, userName: '张三', address: '北京' });
//主键为外部主键时，需要另行指定主键值，此处主键值为 1
store.put({ userName: '张三', address: '北京' }, 1 );
```



Note

当主键为自增、内联主键时不需要指定主键值，当主键为外部主键时，可以将主键值指定为 put()方法的第二个参数值。

提示：在 indexedDB API 中，对象仓库还有一个 add()方法，该方法的使用方法与作用类似于对象仓库的 put()方法。区别在于当使用 put()方法保存数据时，如果指定的主键值在对象仓库中已存在，那么该主键值所在数据被更新为使用 put()方法所保存的数据，而在使用 add()方法保存数据时，如果指定的主键值在对象仓库中已存在，那么保存失败。因此，当出于某些原因只能向对象仓库中追加数据，而不能更新原有数据时，建议使用 add()方法，而 put()方法在其他场合使用。

6.3.7　访问数据

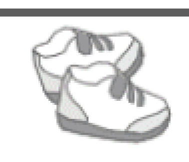


视 频 讲 解

本节介绍如何从 indexedDB 数据库的对象仓库中获取数据。示例完整代码如下所示：

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
function GetData(){
    var dbName = 'indexedDBTest'; //数据库名
    var dbVersion = 20170306; //版本号
    var idb;
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        idb = e.target.result; //引用 IDBDatabase 对象
        var tx = idb.transaction(['Users'],"readonly");
        var store = tx.objectStore('Users');
         var req = store.get(1);
        req.onsuccess = function(){
            if(this.result ==undefined){ alert("没有符合条件的数据"); }
            else{ alert("获取数据成功,用户名为"+this.result.userName); }
        }
        req.onerror = function(){alert("获取数据失败"); }
    };
    dbConnect.onerror = function(){alert('数据库连接失败'); };
}
</script>

<input type="button" value="获取数据" onclick="GetData();"/>
```

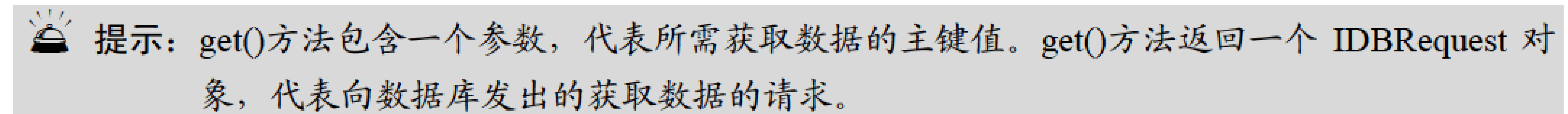


【代码解析】

第 1 步，连接某个 indexedDB 数据库，并且在连接成功后使用该数据库对象的 transaction()方法开启一个只读事务，同时使用 transaction()方法返回的被开启的事务对象的 objectStore()方法获取该事务对象的作用范围中的某个对象仓库。

第 2 步，在获取仓库成功后，可以使用对象仓库的 get()方法从对象仓库中获取一条数据。

Note



提示：get()方法包含一个参数，代表所需获取数据的主键值。get()方法返回一个 IDBRequest 对象，代表向数据库发出的获取数据的请求。

第 3 步，该请求发出后将被立即异步执行，可以通过监听请求对象的 onsuccess 事件（请求执行成功时触发）和请求对象的 onerror 事件（请求执行失败时触发），并指定事件处理函数来定义请求被执行成功或失败时所要进行的处理。

第 4 步，在获取对象的请求执行成功后，如果没有获取到符合条件的数据，此处该数据的主键值为 1，那么该请求对象的 result 属性值为 undefined，如果获取到符合条件的数据，那么请求对象的 result 属性值为获取到的数据记录。在本示例中，指定没有获取到主键值为 1 的数据会弹出提示信息框，提示用户没有获取到该数据记录，否则在弹出提示信息框中显示该数据的 userName（用户名）属性值。

视频讲解

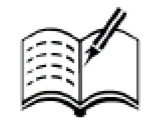
6.3.8 访问键值

通过对象仓库或索引的 get()方法，只能获取到一条数据。在需要通过某个检索条件来检索一批数据时，需要使用 indexedDB API 中的游标。

下面示例设计根据数据记录的主键值检索数据，示例完整代码如下所示。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
var dbName = 'indexedDBTest'; //数据库名
var dbVersion = 20170306; //版本号
var idb;
function window_onload(){
    document.getElementById("btnSaveData").disabled=true;
    document.getElementById("btnSearchData").disabled=true;
}
function ConnectDataBase(){
    /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
    var dbConnect = indexedDB.open(dbName, dbVersion);
    dbConnect.onsuccess = function(e){//连接成功
        idb = e.target.result; //引用 IDBDatabase 对象
        alert('数据库连接成功');
        document.getElementById("btnSaveData").disabled=false;
    };
    dbConnect.onerror = function(){alert('数据库连接失败'); };
}
function SaveData(){
    var tx = idb.transaction(['Users'],"readwrite"); //开启事务
```






```
        tx.oncomplete = function(){alert('保存数据成功');document.getElementById("btnSearchData").disabled= false;}
        tx.onabort = function(){alert('保存数据失败'); }
        var store = tx.objectStore('Users');
        var value = {
            userId: 1,
            userName: '甲',
            address: '北京'
        };
        store.put(value);
        var value = {
            userId: 2,
            userName: '乙',
            address: '上海'
        };
        store.put(value);
        value = {
            userId: 3,
            userName: '丙',
            address: '广州'
        };
        store.put(value);
        value = {
            userId: 4,
            userName: '丁',
            address: '深圳'
        };
        store.put(value);
    }
    function SearchData(){
        var tx = idb.transaction(['Users'],"readonly");
        var store = tx.objectStore('Users');
        var range = IDBKeyRange.bound(1,4);
        var direction = "next";
        var req = store.openCursor(range, direction);
        req.onsuccess = function(){
            var cursor = this.result;
            if(cursor){
                alert('检索到一条数据，用户名为'+cursor.value.userName);
                cursor.continue(); //继续检索
            }else{ alert('检索结束'); }
        }
        req.onerror = function(){alert('检索数据失败'); }
    }
    </script>

    <body onload="window_onload()">
    <input id="btnConnectDataBase" type="button" value="连接数据库" onclick="ConnectDataBase();"/>
    <input id="btnSaveData"    type="button" value="保存数据" onclick="SaveData();"/>
    <input id="btnSearchData" type="button" value="检索数据" onclick="SearchData();"/>
    </body>
```



代码解析：

本示例页面中有三个按钮，分别为“连接数据库”“保存数据”“检索数据”按钮。在页面打开时通过 window.onload 事件函数指定“保存数据”和“检索数据”按钮为无效状态。

用户单击“连接数据库”按钮时执行 ConnectDataBase()函数，在该函数中连接数据库，在数据库连接成功后设定“保存数据”按钮为有效状态。用户单击“保存数据”按钮后会在 Users 对象仓库中保存 4 条数据，数据保存成功后设定“检索数据”按钮为有效状态。

用户单击“检索数据”按钮后，执行 SearchData()函数。在该函数中，通过游标来检索主键值为 1～4 的数据，并将检索到数据的 userName（用户名）属性值显示在弹出的提示信息窗口中。

在 SearchData()函数中使用当前连接的数据库对象的 transaction()方法开启一个只读事务，并且使用 transaction()方法返回的被开启的事务对象的 objectStore()方法获取 Users 对象仓库。

然后，通过对象仓库的 openCursor()方法创建并打开一个游标，该方法有两个参数，其中第一个参数为一个 IDBKeyRange 对象。第二个参数 direction 用于指定游标的读取方向，参数值为一个在 indexedDB API 中预定义的常量值。

openCursor()方法返回一个 IDBRequest 对象，代表一个向数据库发出的检索数据的请求。

调用该方法后立即异步执行，可以通过监听请求对象的 onsuccess 事件（检索数据的请求执行成功时触发），以及请求对象的 onerror 事件（检索数据的请求执行失败时触发），并指定事件处理函数来指定检索数据成功与失败时所执行的处理。

在检索成功后，如果不存在符合检索条件的数据，那么请求对象的 result 属性值为 null 或 undefined，检索终止。可通过判断该属性值是否为 null 或 undefined 来判断检索是否终止并指定检索终止时的处理。

如果存在符合检索条件的数据，那么请求对象的 result 属性值为一个 IDBCursorWithValue 对象，该对象的 key 属性值中保存了游标中当前指向的数据记录的主键值，该对象的 value 属性值为一个对象，代表该数据记录。可通过访问该对象的各个属性值来获取数据记录的对应属性值。

当存在符合检索条件的数据时，可通过 IDBCursorWithValue 对象的 update 方法更新该条数据。可通过 IDBCursorWithValue 对象的 delete()方法删除该条数据。

当存在符合检索条件的数据时，可通过 IDBCursorWithValue 对象的 continue()方法读取游标中的下一条数据记录。

当游标中的下一条数据记录不存在时，请求对象的 result 属性值变为 null 或 undefined，检索终止。

6.3.9　访问属性

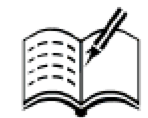


视频讲解

在 indexedDB API 中，可以将对象仓库的索引属性值作为检索条件来检索数据。下面看一个完整示例：

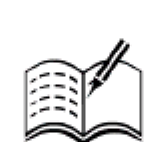
```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
var dbName = 'indexedDBTest'; //数据库名
var dbVersion = 20170306; //版本号
var idb;
function window_onload(){
```






```
        document.getElementById("btnSaveData").disabled=true;
        document.getElementById("btnSearchData").disabled=true;
    }
    function ConnectDataBase(){
        /*连接数据库，dbConnect 对象为一个 IDBOpenDBRequest 对象，代表数据库连接的请求对象*/
        var dbConnect = indexedDB.open(dbName, dbVersion);
        dbConnect.onsuccess = function(e){//连接成功
            idb = e.target.result; //引用 IDBDatabase 对象
            alert('数据库连接成功');
            document.getElementById("btnSaveData").disabled=false;
        };
        dbConnect.onerror = function(){
            alert('数据库连接失败');
        };
    }
    function SaveData(){
        //开启事务
        var tx = idb.transaction(['newUsers'],"readwrite");
        tx.oncomplete = function(){
            alert('保存数据成功');
            document.getElementById("btnSearchData").disabled=false;
        }
        tx.onabort = function(){alert('保存数据失败'); }
        var store = tx.objectStore('newUsers');
        var value = {
            userId: 1,
            userName: '甲',
            address: '北京'
        };
        store.put(value);
        var value = {
            userId: 2,
            userName: '乙',
            address: '上海'
        };
        store.put(value);
        value = {
            userId: 3,
            userName: '丙',
            address: '广州'
        };
        store.put(value);
        value = {
            userId: 4,
            userName: '丁',
            address: '深圳'
        };
        store.put(value);
    }
```






```
function SearchData(){
    //开启事务
    var tx = idb.transaction(['newUsers'],"readonly");
    var store = tx.objectStore('newUsers');
    var idx = store.index('userNameIndex');
    var range = IDBKeyRange.bound('甲','丁');
    var direction = "next";
    var req = idx.openCursor(range, direction);
    req.onsuccess = function(){
        var cursor = this.result;
        if(cursor){
            alert('检索到一条数据,用户名为'+cursor.value.userName);
            cursor.continue(); //继续检索
        }else{ alert('检索结束'); }
    }
    req.onerror = function(){ alert('检索数据失败'); }
}
</script>

<body onload="window_onload()">
<input id="btnConnectDataBase" type="button" value="连接数据库" onclick="ConnectDataBase();"/>
<input id="btnSaveData"    type="button" value="保存数据" onclick="SaveData();"/>
<input id="btnSearchData" type="button" value="检索数据" onclick="SearchData();"/>
</body>
```

在示例页面中共有三个按钮，分别为“连接数据库”“保存数据”“检索数据”按钮。在页面打开时通过 window.onload 事件处理函数指定“保存数据”和“检索数据”按钮为无效状态，在单击“连接数据库”按钮时执行 ConnectDataBase()函数，在该函数中连接数据库，连接成功后设定“保存数据”按钮为有效状态。单击“保存数据”按钮后在 Users1 对象仓库中保存 4 条数据，这 4 条数据的 userName 属性值分别为“甲”“乙”“丙”“丁”。保存成功后设定“检索数据”按钮为有效状态。

单击“检索数据”按钮后执行 SearchData()函数。在该函数中，通过游标来检索 userNameIndex 索引所使用的 userName 属性值为“甲”“丁”的数据，并将检索到数据的 userName（用户名）属性值显示在弹出的提示信息窗口中。

在 SearchData()函数中，使用当前连接的数据库对象的 transaction()方法开启一个只读事务，并且使用 transaction()方法返回的被开启的事务对象的 objectStore()方法获取 newUsers 对象仓库，同时使用对象仓库的 index()方法获取 userNameIndex 索引。

最后，需要通过索引的 openCursor()方法创建并打开一个游标。

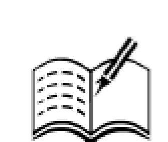

视频讲解

6.4　案例：设计录入表单

本例使用 indexedDB API 设计一个电子刊物发布的应用，演示效果如图 6.14 所示。在示例页面中显示 3 个表单框，第一个表单框登录电子刊物信息，并保存在 indexedDB 数据库中。第二个表单框用于管理数据库中的记录，可以根据需要清空全部记录，或者删除指定的记录。第三个表单框用于显示数据库中所有的电子刊物记录。









线 上 阅 读

图 6.14　电子刊物发布应用效果

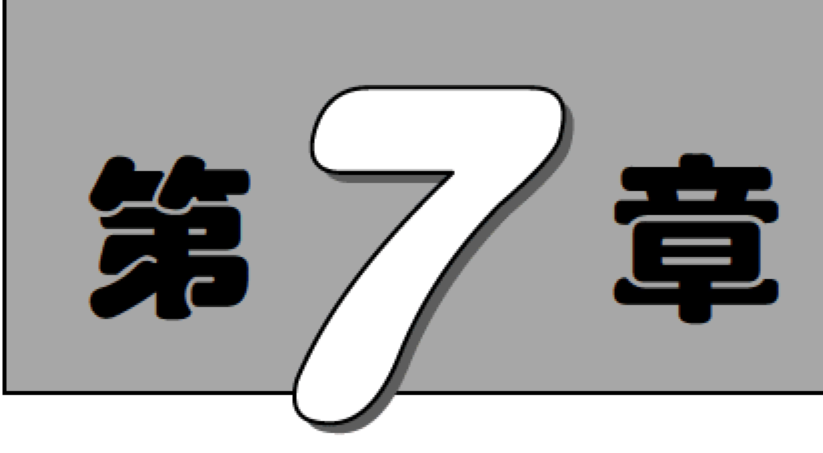
具体操作步骤请扫码学习。

6.5　在 线 练 习

练习使用 HTML5 本地存储方法。



在 线 练 习



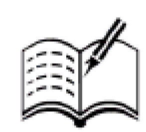
第7章 应用程序缓存

HTML5 新增 ApplicationCache API 接口，提供了应用程序缓存的功能。在页面加载时，允许用户缓存各种资源文件。这样在离线状态下，应用程序可以读写缓存文件，不需要与远程保持联系；当在线状态时，应用程序能够根据缓存文件及时更新远程数据，保证缓存数据与远程数据同步；如果缓存文件与远程文件同步，则优先访问缓存文件，以提升访问速度和运行效率。

【学习重点】

- 正确使用 manifest 文件。
- 使用 HTML5 ApplicationCache API 设计 Web 离线应用程序。





7.1　ApplicationCache API 基础

HTML5 通过 ApplicationCache API 使离线存储成为可能。离线存储（Offline Storage）的核心功能：在断网状态下，用户依然能够访问站点；在联网状态下，会自动更新缓存数据。利用 HTML5 离线存储功能可以开发出很多丰富的基于 Web 的应用。

7.1.1　认识 ApplicationCache API

一个页面刚加载完毕，突然断网，刷新页面后就没了。如果刷新页面后，还是刚才的页面，在新窗口中重新访问该页面，输入相同的网址，在断网状态下打开依然是原来那个页面。如果在没有网络的地方（如飞机上）和时候（网络坏了），也能够进行 Web 操作，等到有网络的时候，再同步到 Web 上，就大大方便了用户的使用。

越来越多的应用被移植到云端，但网络连接中断时有发生，如外出旅行、身处无网环境等。间断性的网络连接一直是网络计算系统致命的弱点，如果应用程序完全依赖于与网络的通信，而网络又无法连接时，用户就无法正常使用应用程序。

HTML5 的 ApplicationCache API 综合了 Web 应用和桌面应用两者的优势，基于 Web 技术构建的 Web 应用程序，可以在浏览器中运行并在线更新，也可在脱机情况下使用。离线应用缓存使得在无网络连接状态下运行应用程序成为可能，这类应用程序用处很多，简单举例说明如下：

☑　阅读和撰写电子邮件。

☑　编辑文档。

☑　编辑和显示演示文档。

☑　创建待办事宜列表。

HTML5 离线应用有三个好处：

☑　用户可以离线访问 Web 应用，不用时刻保持与互联网的连接。

☑　因为文件被缓存在本地，提升了页面加载速度。

☑　离线应用只加载被修改过的资源，因此大大降低了用户请求服务器造成的负载压力。

用户可以直接控制应用程序缓存。利用缓存清单文件将相关资源组织到同一个逻辑应用中。这样 Web 应用就拥有了桌面应用的特性。缓存清单文件中标识的资源构成了应用缓存（Application Cache），它是浏览器持久性存储资源的地方，通常在硬盘上。有些浏览器向用户提供了查看应用程序缓存中数据的方法。

例如，在新版 Firefox 中，about:cache 页面会显示应用程序缓存的详细信息，提供了查看缓存中的每个文件的方法，如图 7.1 所示。

浏览器对 HTML5 离线应用的支持情况如表 7.1 所示，从中可以看到目前大部分浏览器已经支持 HTML5 离线应用。

HTML5 离线应用的支持程度不同，在使用之前建议先测试浏览器的支持情况。检测方法如下：

```
if(window.applicationCache) {
    //浏览器支持的离线应用
}
```




Note

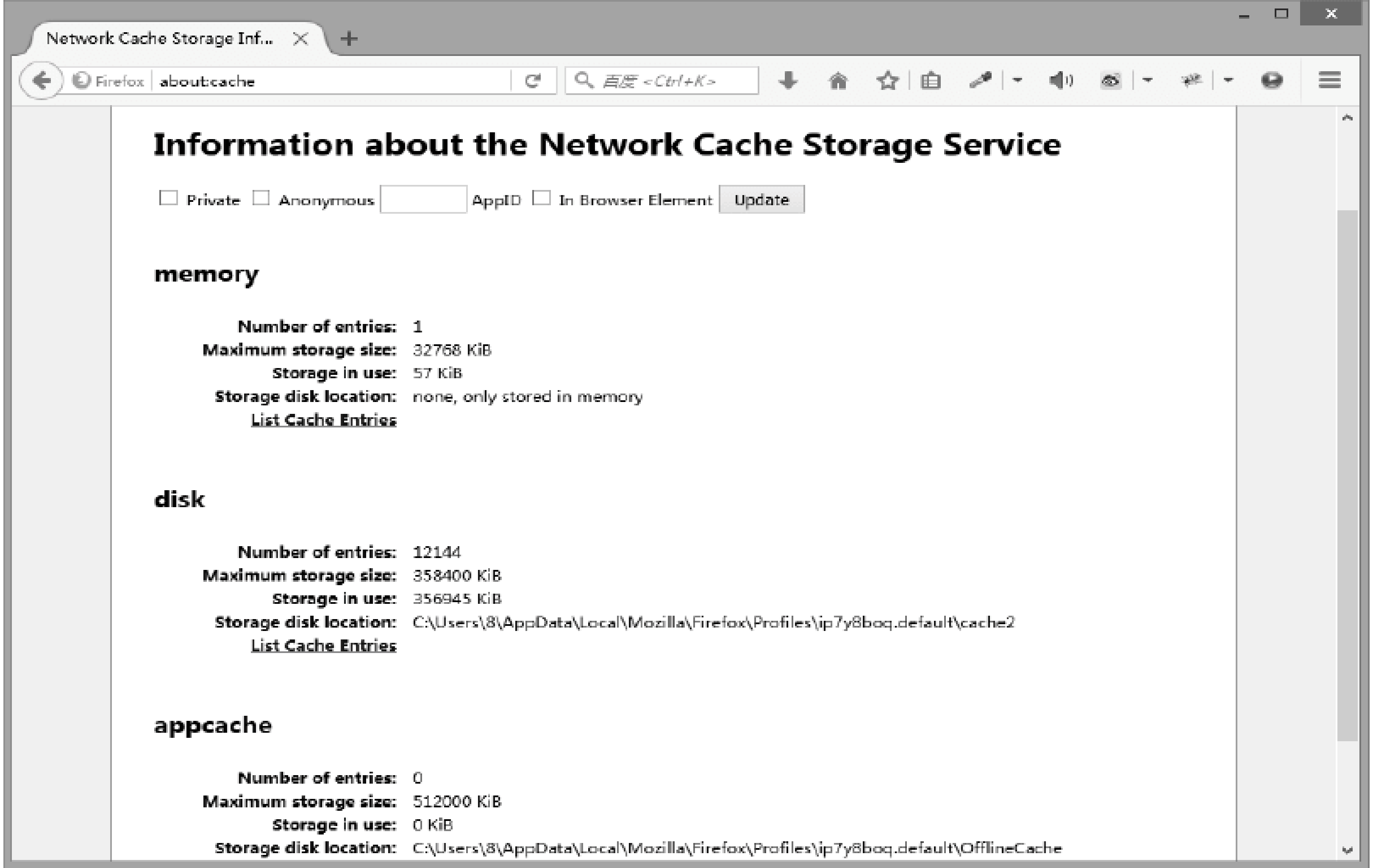


图 7.1　Firefox 的 about:cache 页面

表 7.1　浏览器支持概述

浏　览　器	说　　明
IE	不支持
Firefox	3.5 及以上的版本支持
Opera	10.6 及以上的版本支持
Chrome	4.0 及以上的版本支持
Safari	4.0 及以上的版本支持
iPhone	2.0 及以上的版本支持
Android	2.0 及以上的版本支持

7.1.2　配置服务器

HTML5 离线缓存包含两部分内容：

☑　manifest 文件：manifest 文件包含了需要缓存的资源清单。

☑　JavaScript 程序：提供用于更新缓存文件的方法，以及对缓存文件的操作。

manifest 文件是一个文本文件，列出了浏览器为离线应用缓存的所有资源。manifest 文件的 MIME 类型是 text/cache-manifest。

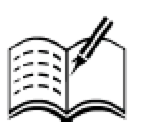
Python 标准库中的 SimpleHTTPServer 模块对扩展名为.manifest 的文件能配以头部信息：Content-type:text/cache-manifest，配置方法是打开 PYTHON_HOME/Lib/mimetypes.py 文件并添加一行代码：

```
'. manifest': 'text/cache-manifest manifest',
```

如果要配置 Apache HTTP 服务器，需要将下面一行代码添加到 Apache Software Foundation\Apache2.2\conf 文件夹的 mime.type 文件中，如图 7.2 所示。

```
text/cache-manifest manifest
```





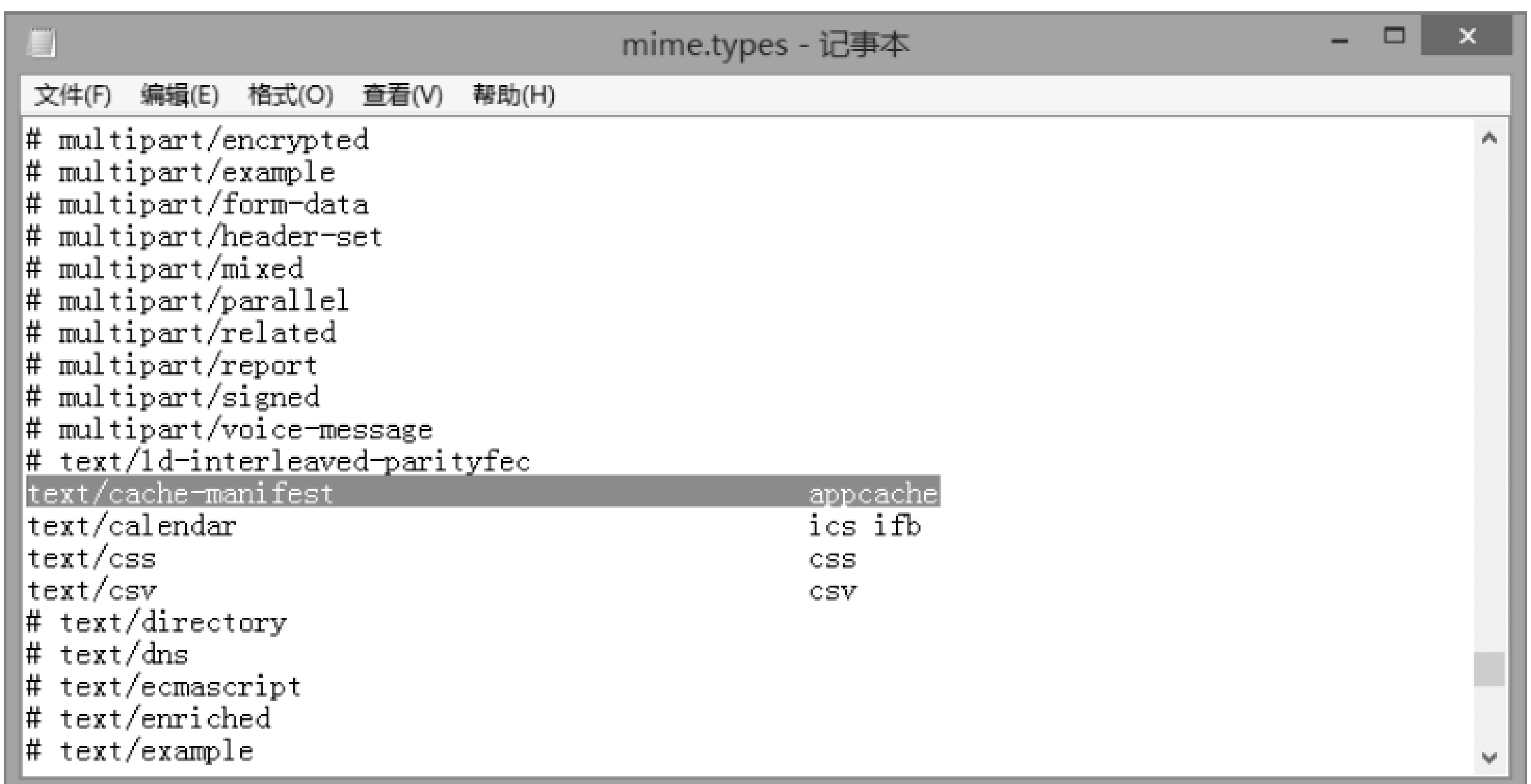


图 7.2　配置 Apache HTTP 服务器



7.1.3　认识 manifest

manifest 文件的基本语法如下：

☑ 第一行必须以 CACHE MANIFEST 字符串开头。

☑ 然后换行，每行单列资源文件（包含路径）。

☑ 每行的换行符可以是 CR、LF 或者 CRLF。

☑ 文本编码格式必须是 UTF-8。

☑ 注释必须以#开头。

manifest 文件包括三个节点：

☑ CACHE:

manifest 文件的默认入口，在此入口之后罗列的文件，或直接写在 CACHE MANIFEST 后的文件在下载到本地后会被缓存起来。

☑ NETWORK:

可选节点，在此节点后面所罗列的文件是需要访问网络的，即使用户离线访问，也会直接跳过缓存而访问服务器。

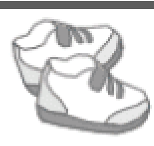
☑ FALLBACK:

可选节点，指定无法访问资源时的回调页面。每一行包括两个 URI，第一个是资源文件 URI，第二个是回调页面 URI。

注意：以上节点没有先后顺序，而且在同一个 manifest 中可以多次出现。

【示例 1】新建一个以 manifest 为扩展名的文件，命名为 cacheData.manifest，在这个文件中设置要缓存的文件路径列表：login.html、i.css、alipay-i-logo-big.png、alipay-i-icons.png、mui-min.js；另外定义需要访问网络的文件列表：button-ok.png，以及备用文件列表：alipay-bank-cmb.png。

```
CACHE MANIFEST
#version 1.0
login.html
static/css/i.css
static/img/png/alipay-i-logo-big.png
```



```
static/img/png/alipay-i-icons.png
static/js/mui-min.js
NETWORK:
static/img/png/button-ok.png
CACHE:
static/img/png/login-slider-bg.png
FALLBACK:
static/img/png/alipay-bank-icbc.png static/img/png/alipay-bank-cmb.png
```

每个站点都有 5MB 空间来存储缓存数据，如果 manifest 文件或文件里所列的文件无法加载，整个缓存更新过程将无法进行，浏览器会使用最后一次成功的缓存数据。

如果没有指定“CACHE:”节点标题，那么默认就是 CACHE MANIFEST 部分。下面 manifest 文件设置了两个要缓存的文件：

```
CACHE MANIFEST
application.js
style.css
```

添加到 CACHE MANIFEST 区块中的文件，无论应用程序是否在线，浏览器都会从应用程序缓存中获取该文件。没有必要在这里列出应用程序的主 HTML 资源，因为最初指向 manifest 文件的 HTML 文档会被隐含进来。但是，如果希望缓存多个 HTML 文件，或者希望将多个 HTML 文件作为支持缓存的应用程序的可选入口，则需将这些文件列在 CACHE MANIFEST 中。

FALLBACK 部分提供了获取不到缓存资源时的备选资源路径。第一个文件的路径和第二个文件的路径中间有一个空格，这个“FALLBACK:”的作用是：当第一个文件缓存不成功，或无法找到时，它会缓存第二个文件。

【示例 2】下面示例设计当无法获取 app/ajax 时，所有 app/ajax/及其子路径的请求都会被转发给 default.html 文件来处理。

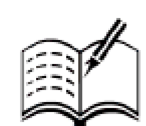
```
CACHE MANIFEST
#缓存文件列表
about.html
html5.css
index.html
happy-trails-rc.gif
lake-tahoe.JPG
#不缓存注册页面
NETWORK
signup.html
FALLBACK
signup.html offline.html
/app/ajax/ default.html
```

【示例 3】#表示注释行标识符，但它还有一个小作用：Web 应用的缓存只有在 manifest 文件被修改的情况下才会被更新，所以如果只是修改了被缓存的文件，那么用户本地的缓存还是不会被更新的，但是可以通过修改 manifest 文件来告诉浏览器需要更新缓存。

```
CACHE MANIFEST
# wanz app v1

#指明缓存入口
```






```
CACHE:
index.html
style.css
images/logo.png
scripts/main.js

#以下资源必须在线访问
NETWORK:
login.php

#如果 index.php 无法访问则用 404.html 代替
FALLBACK:
/index.php /404.html
```

修改注释行的文件版本：

```
# wanz app v2
```

这样做有三个好处：

☑　可以很明确地了解离线 Web 应用的版本。

☑　通过简单地修改这个版本号就可以轻易地通知浏览器进行更新。

☑　可以配合 JavaScript 程序来完成缓存更新。

【示例 4】创建好了 cacheData.manifest 文件，下面就需要在 HTML 文件中指定文档的 manifest 属性为 cache.mnifest 文件的路径。

```
<html manifest="cacheData.manifest"></html>
```

manifest 的文件路径可以是绝对路径或相对路径，甚至可以引用其他服务器上的 manifest 文件。该文件所对应的 mime-type 应该是 text/cache-manifest，所以需要配置服务器来发送对应的 MIME 类型信息。

引用 manifest 文件的页面，不管有没有罗列清单，都会被缓存。

提示：由于有某些浏览器中仅仅添加这一属性，可能并不能很好地工作，所以一定要用 HTML5 文档声明方式创建 HTML 页面。

```
<!DOCTYPE html>
<html manifest="cacheData.manifest"></html>
```

7.1.4　使用 ApplicationCache



使用 ApplicationCache 需要三步操作：

第 1 步，配置服务器 manifest 文件的 MIME 类型；

第 2 步，编写 manifest 文件；

第 3 步，在页面的 html 元素的 manifest 属性中引用 manifest 文件。

完成上述三步之后，即使拔掉网线，也可以访问页面。

注意：启用离线应用之后，当修改 JavaScript 代码或 CSS 样式，然后将更新内容上传到服务器，在本地刷新页面重新预览时，会发现无法看到最新的页面效果。那是因为本地浏览器还没有更新 HTML5 的离线存储文件。



更新 HTML5 离线缓存有三种方法：

☑ 清除离线存储的数据。这个不一定是清理浏览器历史记录就可以做到的，因为不同浏览器管理离线存储的方式不同。例如，在 Firefox 浏览器中需要选择“选项”→“高级”→“网络”→“脱机存储”命令，然后在其中清除离线存储数据。

☑ 修改 manifest 文件。修改了 manifest 文件里所罗列的文件也不会更新缓存，而是要更新 manifest 文件。

☑ 使用 JavaScript 编写更新程序。

ApplicationCache API 是离线缓存的应用接口，通过 window.applicationCache 对象可触发一系列与缓存状态相关的事件。该对象有一个数值型属性 window.applicationCache.status，它代表了缓存的状态。缓存状态共有 6 种，说明如表 7.2 所示。

表 7.2　缓存状态说明

Status 值	说　　明
0	UNCACHED（未缓存）
1	IDLE（空闲）
2	CHECKING（检查中）
3	DOWNLOADING（下载中）
4	UPDATEREADY（更新就绪）
5	OBSOLETE（过期）

目前，互联网上大部分的页面都没有指定缓存清单，所以这些页面的状态就是 UNCACHED（未缓存）。IDLE（空闲）是带有缓存清单的应用程序的典型状态。处于空闲状态说明应用程序的所有资源都已被浏览器缓存，当前不需要更新。如果缓存曾经有效，但现在 manifest 文件丢失，则缓存进入 OBSOLETE（过期）状态。对于上述各种状态，API 包含了与之对应的事件和回调特性。

例如，当缓存更新完成进入空闲状态时，会触发 cached 事件。此时，可能会通知用户，应用程序已处于离线模式可用的状态，可以断开网络连接了。如表 7.3 所示是一些与缓存状态有关的常见事件。

表 7.3　缓存事件说明

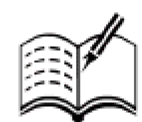
事　　件	说　　明
oncached	IDLE（空闲）
onchecking	CHECKING（检查中）
ondownloading	DOWNLOADING（下载中）
onupdateready	UPDATEREADY（更新就绪）
onobsolete	OBSOLETE（过期）

此外，没有可用更新或者发生错误时，还有一些表示更新状态的事件，例如：

```
onerror
onnoupdate
onprogress
```

window.applicationCache 有一个 update()方法，调用该方法会请求浏览器更新缓存。包括检查新版本的 manifest 文件并下载必要的新资源。如果没有缓存或者缓存已过期，则会抛出错误。





【示例 1】其中常用代码说明如下：

```
//返回应用于当前 window 对象文档的 ApplicationCache 对象
cache = window.applicationCache
//返回应用于当前 shared worker 的 ApplicationCache 对象 [shared worker]
cache = self.applicationCache
//返回当前应用的缓存状态，status 有五种无符号短整型值的状态，说明如表 7.2 所示
cache.status
//调用当前应用资源下载过程
cache.update()
//更新到最新的缓存，该方法不会使之前加载的资源突然被重新加载
cache.swapCache()
```

调用 swapCache()方法，图片不会重新加载，样式和脚本也不会重新渲染或解析。在获取 status 属性时，它返回当前 applicationCache 的状态，它的值有以下几种状态：

☑ UNCACHED (0)：ApplicationCache 对象的缓存宿主与应用缓存无关联。

☑ IDLE (1)：应用缓存已经是最新的，并且没有标记为 obsolete。

☑ CHECKING (2)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 checking。

☑ DOWNLOADING (3)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 downloading。

☑ UPDATEREADY (4)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 idle，并且没有标记为 obsolete，但是缓存不是最新的。

☑ OBSOLETE(5)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 obsolete。

如果 update 方法被调用了，浏览器就必须在后台调用应用缓存下载过程；如果 swapCache 方法被调用了，浏览器会执行以下步骤：

第 1 步，检查 ApplicationCache 的缓存宿主是否与应用缓存关联。

第 2 步，让 cache 成为 ApplicationCache 对象的缓存宿主关联的应用缓存。

第 3 步，如果 cache 的应用缓存组被标记为 obsolete，那么就取消 cache 与 ApplicationCache 对象的缓存宿主的关联并取消这些步骤，此时所有资源都会从网络中下载而不是从缓存中读取。

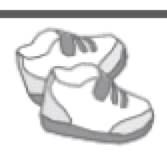
第 4 步，检查同一个缓存组中是否存在完成标志为“完成”的应用缓存，版本比 cache 更新。

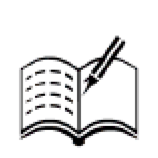
第 5 步，让完成标志为“完成”的新 cache 成为最新的应用缓存。

第 6 步，取消 cache 与 ApplicationCache 对象的缓存宿主的关联并用新 cache 代替关联。

【示例 2】通过下面的代码可以检查当前页面缓存的状态。

```
var appCache = window.applicationCache;
switch (appCache.status) {
    case appCache.UNCACHED:
        // UNCACHED == 0
        alert('UNCACHED');
        break;
    case appCache.IDLE:
        // IDLE == 1
        alert('IDLE');
        break;
```




Note

```
        case appCache.CHECKING:
            // CHECKING == 2
            alert('CHECKING');
            break;
        case appCache.DOWNLOADING:
            // DOWNLOADING == 3
            alert('DOWNLOADING');
            break;
        case appCache.UPDATEREADY:
            // UPDATEREADY == 5
            alert('UPDATEREADY');
            break;
        case appCache.OBSOLETE:
            // OBSOLETE == 5
            alert('OBSOLETE');
            break;
        default:
            alert('UKNOWN CACHE STATUS');
            break;
    };
```

更新的实现过程大概是这样的：

调用 applicationCache.update()让浏览器开始尝试更新。操作前提是 manifest 文件是更新过的，如修改 manifest 版本号。

在 applicationCache.status 为 UPDATEREADY 状态时，就可以调用 applicationCache.swapCache()方法来将旧的缓存更新为新的。

```
var appCache = window.applicationCache;
appCache.update();                //开始更新
if (appCache.status == window.applicationCache.UPDATEREADY) {
    appCache.swapCache();         //得到最新版本缓存列表，且成功下载资源，更新缓存到最新
}
```

提示： 更新过程很简单，但是一个好的应用少不了容错处理，如 Ajax 技术一样，需要对更新过程进行监控，处理各种异常或提示等待状态来使 Web 应用更强壮、用户体验更好，因此需要了解 applicationCache 的更新过程所触发的事件，主要包括 onchecking、onerror、onnoupdate、ondownloading、onprogress、onupdateready、oncached 和 onobsolete。要对更新错误进行处理，可以这样写：

```
var appCache = window.applicationCache;
//请求 manifest 文件时返回 404 或 410，下载失败
//或 manifest 文件在下载过程中源文件被修改会触发 error 事件
appCache.addEventListener('error', handleCacheError, false);
function handleCacheError(e) {
    alert('Error: Cache failed to update!');
};
```

不管是 manifest 文件，还是它所罗列的资源文件下载失败，整个更新过程终止了，浏览器会使用上一个最新的缓存。





视 频 讲 解



Note

7.1.5　事件监听

HTML5 引入了一些新的事件，用来让应用程序检测网络是否正常连接。应用程序处于在线状态和离线状态会有不同的行为模式。是否处于在线状态可以通过检测 window.navigator 对象的属性来做判断。

navigator.onLine 是一个标明浏览器是否处于在线状态的布尔属性。当然，onLine 值为 true 并不能保证 Web 应用程序在用户的机器上一定能访问到相应的服务器。而当其值为 false 时，不管浏览器是否真正联网，应用程序都不会尝试进行网络连接。

【示例 1】查看页面状态是在线还是离线的代码如下：

```
//当页面加载时，设置状态为 online 或者 offline
function loadDemo() {
    if(navigator.onLine) {
        log("Online");
    } else {
        log("Offline");
    }
}
//增加事件监听，当在线状态发生变化时，将触发响应
window.addEventListener("online", function(e) {
    log("Online");
}, true);
window.addEventListener("offline", function(e) {
    log("Offline");
}, true);
```

【示例 2】在支持 HTML5 离线存储的浏览器中，window 对象有一个 applicationcache 属性，通过 window.applicationcach 可以获得一个 DOMApplicationCache 对象，这个对象来自 DOMApplicationCache 类，这个类有一系列的属性和方法。

首先，获取 DOMApplicationCache 对象。

```
var cache = window.applicationcache;
```

接着，触发 cache 对象的一些事件来检测缓存是否成功。

```
/*oncached 事件表示：当更新已经处理完成，并且存储。
 * 如果一切正常，这里 cache 的状态应该是 4
 */
cache.addEventListener('cached', function() {
    console.log('Cached,Status:' + cache.status);
}, false);
/*onchecking 事件表示：当更新已经开始进行，但资源还没有开始下载，意思就是说：刚刚获取到最新的资源。
 * 如果一切正常，这里 cache 的状态应该是 2
 */
cache.addEventListener('checking', function() {
    console.log('Checking,Status:' + cache.status);
}, false);
/*ondownloading 事件表示：开始下载最新的资源。
 * 如果一切正常，这里 cache 的状态应该是 3
 */
```



Note

```
cache.addEventListener('downloading', function() {
    console.log('Downloading,Status:' + cache.status);
}, false);
/*onerror 事件表示：有错误发生，如 manifest 文件找不到，或服务端有错误，或资源找不到，都会触发 onerror 事件。
 * 如果一切正常，这里 cache 的状态应该是 0
 */
cache.addEventListener('error', function() {
    console.log('Error,Status:' + cache.status);
}, false);
/*onnoupdate 事件表示：更新已经处理完成，但是 manifest 文件还未改变，处理闲置状态。
 * 如果一切正常，这里 cache 的状态应该是 1
 */
cache.addEventListener('noupdate', function() {
    console.log('Noupdate,Status:' + cache.status);
}, false);
/*onupdateready 事件表示：更新已经处理完成，新的缓存可以使用。
 * 如果一切正常，这里 cache 的状态应该是 4
 */
cache.addEventListener('updateready', function() {
    console.log('Updateready,Status:' + cache.status);
    cache.swapCache();
}, false);
```

通过以上代码可以发现，当 DOMApplicationCache 对象触发了 updateready 事件时，才真正更新了缓存文件。

如果在开发过程中就开始对离线存储功能做单元测试，那么每一次修改文件都必须要更新 manifest 文件中的内容，即使更新了一个注释，整个 manifest 文件也会更新，DOMApplicationCache 对象也会触发上述的一系列事件，直到新的缓存文件可用为止。通常情况下，一般都是通过更新 manifest 文件中的版本号以触发 onupdateready 事件。

7.2 案例实战

下面通过几个具体案例熟悉 HTML5 的离线缓存的具体应用。

视频讲解

7.2.1 缓存首页

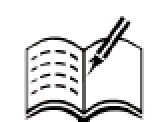
本节示例将通过一个简单的首页缓存演示 HTML 离线缓存的应用。整个过程只需要简单的 5 步即可完成，当然要设计更加复杂的离线应用，还需要读者结合 HTML5 其他新技术，并进行更加复杂的设置才行。

【操作步骤】

第 1 步，添加 HTML5 Doctype。创建符合规范的 HTML5 文档。HTML5 Doctype 相比于 XHTML 版本的 doctype 而言，要简单明了很多。

```
<link href="style.css" type="text/css" rel="stylesheet" media="screen">
```





```
<meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0;">

<div id="container">
    <header class="ma-class-en-css">
        <h1 id ="logo"><a href="#">HTML5</a></h1>
    </header>
    <div id="content">
        <h2>HTML5</h2>
        <p>HTML 标准自 1999 年 12 月发布的 HTML 4.01 后，后继的 HTML 5 和其他标准被束之高阁，为了推动 web 标准化运动的发展，一些公司联合起来，成立了一个叫作 Web Hypertext Application Technology Working Group （Web 超文本应用技术工作组 - WHATWG）的组织，HTML5 草案的前身名为 Web Applications 1.0，于 2004 年被 WHATWG 提出，于 2007 年被 W3C 接纳，并成立了新的 HTML 工作团队。</p>
        <p>HTML 5 的第一份正式草案已于 2008 年 1 月 22 日公布。HTML 5 有两大特点：首先，强化了 Web 网页的表现性能。其次，追加了本地数据库等 Web 应用的功能。</p>
    </div>
    <footer>html5 by <a href="#">WHATWG</a></footer>
</div>
```

然后另存为 index1.html，放在站点根目录下。

第 2 步，添加.htaccess 支持。在创建用于缓存页面的 manifest 清单文件之前，先要在.htaccess 文件中添加以下代码，具体说明请参考上节说明。

```
AddType text/cache-manifest .manifest
```

该指令可以确保每个 manifest 文件为 text/cache-manifest MIME 类型。如果 MIME 类型不对，那么整个清单将没有任何效果，页面将无法离线应用。

> **注意**：本章案例都在 Apache HTTP Server 服务器环境下运行，读者在测试之前，应该在本地计算机中构建虚拟的 Apache 服务器环境。

> **提示**：htaccess 文件被称为分布式配置文件，是 Apache 服务器中的一个配置文件，它提供了针对目录改变配置的方法，负责相关目录下的网页配置。通过 htaccess 文件，可以帮我们实现：网页重定向、自定义错误页面、改变文件扩展名、允许/阻止特定的用户或者目录的访问、禁止目录列表、配置默认文档等功能。

启用.htaccess，需要修改 httpd.conf，启用 AllowOverride，并可以用 AllowOverride 限制特定命令的使用。如果需要使用.htaccess 以外的其他文件名，可以用 AccessFileName 指令来改变。例如，需要使用.config，则可以在服务器配置文件中按以下方法配置：AccessFileName.config。

第 3 步，创建 manifest 文件。配置服务器之后，就可以创建 manifest 清单文件。新建一个文本文档，另存名为 offline.manifest，然后输入以下代码。

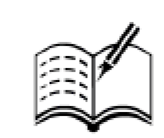
```
CACHE MANIFEST
#This is a comment

CACHE:
index.html
style.css
image.jpg
image-med.jpg
```



```
image-small.jpg
notre-dame.jpg
```

在 CACHE 声明之后，罗列出所有需要缓存的文件。这对于缓存简单页面来说已经足够。但是HTML5 缓存还有更多可能。例如，考虑以下 manifest 文件：



Note

```
CACHE MANIFEST
#This is a comment

CACHE:
index.html
style.css

NETWORK:
search.php
login.php

FALLBACK:
/api offline.html
```

其中 CACHE 声明用于缓存 index.html 和 style.css 文件。同时，NETWORK 声明用于指定无须缓存的文件，如登录页面。最后一个是 FALLBACK 声明，这个声明允许在资源不可用的情况下，将用户重定向到特定文件，如 offline.html。

第 4 步，关联 manifest 文件到 HTML 文档。设计完 manifest 文件和 HTML 文档，还需要将 manifest 文件关联到 HTML 文档中。使用 html 元素的 manifest 属性：

```
<html manifest="/offline.manifest">
```

第 5 步，测试文档。完成后，使用 Firefox 3.5+本地访问 index.html 文件，效果如图 7.3 所示，浏览器会默认自动缓存。



图 7.3　测试首页离线缓存

此后即使服务器停止工作或者无法上网，我们依然可以访问服务器上的该首页。如果没有离线存储的支持，则当服务器停止工作或者无法上网时访问首页，将会显示如图 7.4 所示的效果。



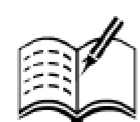






图 7.4　不支持离线缓存的效果



7.2.2　离线编辑

本例利用 HTML5 ApplicationCache API，通过在线状态检测，并配合 DOM Storage 等功能来设计一个离线应用。设计思路：开发一个便签管理的 Web 应用程序，用户可以在其中添加和删除便签。它支持离线功能，允许用户在离线状态下添加、删除便签，并且当在线以后能够同步到服务器上。



具体操作步骤和代码解析请扫码学习。

7.3　在 线 练 习

练习使用 HTML5 应用缓存的功能。



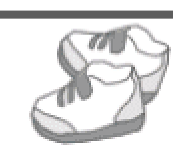
第8章

多线程编程

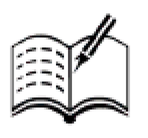
JavaScript是单线程编程语言，运算能力比较弱。HTML5新增的Web Workers API能够创建一个不影响前台处理的后台线程，并且在这个后台线程中可以继续创建多个子线程，以帮助JavaScript实现多线程运算的能力。

【学习重点】

- 创建线程对象。
- 使用Web Workers通信。
- 设计多线程编程页面。



8.1　Web Workers 基础



通过 Web Workers，用户可以将耗时较长的处理交给后台线程去运行，从而解决了 HTML5 之前因为某个处理耗时过长而出现的“假死”现象。

8.1.1　认识 Web Workers

HTML5 新增 Web Workers API，该 API 可以帮助用户创建在后台运行的线程，这个线程被称为 worker，如果将可能耗费较长时间的处理交给后台去执行的话，对用户在前台页面中执行的操作就没有影响。

尽管 Web Workers 功能强大，但也不是万能的，有些事情它还做不到。例如，在 Web Workers 中执行的脚本不能访问该页面的 window 对象，因此 Web Workers 不能直接访问 Web 页面和 DOM API，虽然 Web Workers 不会导致浏览器 UI 停止响应，但是仍然会消耗 CPU 周期，导致系统反应速度变慢。

如果开发人员创建的 Web 应用程序需要执行一些后台数据处理，但又不希望这些数据处理任务影响 Web 页面本身的交互性，那么可以通过 Web Workers 生成一个 Web Worker 去执行数据处理任务。同时添加一个事件监听器进行监听，并与之进行数据交互。

Web Workers 另一个用途：监听后台服务器广播的消息，收到后台服务器的消息后，将其显示在 Web 页面上。这种与后台服务器对话的场景，可能会用到 Web Sockets 或 Server-Sent 事件。

Web Workers 可执行的操作：

加载一个 JavaScript 文件，进行大量的复杂计算，而不挂起主进程，并通过 postMessage，onmessage 进行通信。

可以在 worker 中通过 importScripts(url)方法加载 JavaScript 脚本文件。

可以使用 setTimeout()、clearTimeout()、setInterval()和 clearInterval()。

可以使用 XMLHttpRequest 进行异步请求。

可以访问 navigator 的部分属性。

可以使用 JavaScript 核心对象。

Web Workers 不可执行的操作：

不能跨域加载 JavaScript。

Worker 内代码不能访问 DOM。如果改变 DOM，只能通过返回消息给创建者的回调函数进行处理。

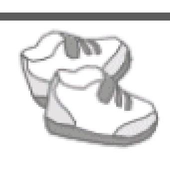
各个浏览器对 Worker 的实现还没有完全完善。不是每个浏览器都支持所有新特性。

使用 Web Workers 加载数据没有 JSONP 和 Ajax 加载数据高效。

各浏览器对 HTML5 Web Workers 的支持情况如表 8.1 所示，从中可以看到目前浏览器对 Web Workers 的支持情况各不相同，但 Web Workers 已经得到了大部分主流浏览器的支持，并且仍在持续更新发展中。

表 8.1　浏览器支持概述

浏　览　器	说　　明
IE	不支持





Note

续表

浏 览 器	说 明
Firefox	3.5 及以上的版本支持
Opera	10.6 及以上的版本支持
Chrome	3.0 及以上的版本支持
Safari	4.0 及以上的版本支持

在调用 Web Workers API 函数之前。应该确认当前浏览器是否支持。如果不支持，可以提供一些备用信息，提醒用户使用最新的浏览器。

【示例】下面代码可以用来测试浏览器是否支持。

```
function testWorker() {
    if (typeof(Worker) !== "undefined") {
        document.getElementById("support").innerHTML = "浏览器支持 HTML5 Web Workers";
    }
}
```

在上面代码中，使用 testWorker()函数来检测浏览器的支持情况，可在页面加载时调用该函数。调用 typeof(Worker)会返回全局 window 对象的 Worker 属性，如果浏览器不支持 Web Workers API，则返回结果将是 undefined。上面代码在检测了浏览器支持性之后，会将检测结果反馈到页面上。

视 频 讲 解

8.1.2 创建 Web Workers

调用 Worker 构造函数可以创建一个 worker（线程）。Workers 在初始化时会接收一个 URL 参数，参数 URI 表示要执行的脚本文件地址，其中包含了供 Worker 执行的代码。

```
worker = new Worker("echoWorker.js");
```

当创建 Worker 对象后，Web Workers 就可以通过 postMessage()方法向任务池发送任务请求，执行完之后再通过 postMessage()返回消息给创建者指定的事件处理程序，然后通过 onmessage 捕获返回消息，实现前后台数据的交互。

【示例 1】如果获取 worker 进程的返回值，可以通过 onmessage 事件处理程序进行监听。

```
var myWorker = new Worker('easyui.js');
myWorker.onmessage = function(event){
    alert('Called back by the worker!');
};
```

在上面代码中，第一行代码将创建和运行 worker 进程，第二行设置 worker 的 onmessage 属性，绑定事件，当 worker 的 postMessage()方法被调用时，这个被绑定的函数就会被调用。

对于由多个 JavaScript 文件组成的应用程序来说，可以通过包含 script 元素的方式，在页面加载的时候同步加载 JavaScript 文件。由于 Web Workers 没有访问 document 对象的权限，所以 Worker 只能使用 importScripts()方法导入其他 JavaScript 文件。

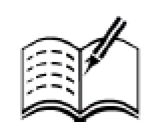
importScripts()是全局函数，该函数可以将脚本或库导入它们的作用域中，导入的 JavaScript 文件只会在某一个已有的 Worker 中加载和执行。多个脚本的导入同样也可以使用 importScripts()函数，它们会按顺序执行。

【示例 2】importScripts()可以接收空的参数或多个脚本 URI 参数，下面形式都是合法的：



```
importScripts();
importScripts('foo.js');
importScripts('foo.js','bar.js');
```

JavaScript 会加载列出每一个脚本文件，然后运行并初始化。这些脚本中的任何全局对象都可以被 worker 使用。



注意：importScripts()方法下载脚本顺序可能不一样，但执行的顺序一定是按 importScripts()方法中列出的顺序进行，而且是同步的，在所有脚本加载完并运行结束后 importScripts()才会返回。

Web Workers 能够嵌套使用，以创建子 Worker：

```
var subWorker = new Worker("subWorker.js");
```

用户可以创建多个 workers。子 worker 必须寄宿于同一个父页面下，且它的 URI 必须与父 worker 的地址同源，这样可以很好地维持它们的依赖关系。

Web Workers 可以使用 setTimeout()和 setInterval()。如果希望 Web Workers 进程周期性地运行而不是不停地循环下去的话，使用这两个方法非常有用。

注意：在后台线程中是不能访问页面或窗口对象，此时如果在后台线程的脚本文件中使用 window 对象或 document 对象，则会引发错误。

【示例 3】用户可以通过 Worker 对象的 onmessage 事件获取后台线程反馈的消息。

```
worker.onmessage=function( event){
    //处理收到的消息
}
```

使用 Worker 对象的 postMessage()方法可以给后台线程发送消息。发送的消息是文本数据，但也可以是任何 JavaScript 对象，需要通过 JSON 对象的 stringify()方法将其转换成文本数据。

```
worker.postMessage(meseage);
```

通过获取 Worker 对象的 onmessage 事件句柄及 Worker 对象的 postMessage()方法可以实现线程内部的消息接收和发送。

【拓展】

在使用 Web Workers 之前，用户应该熟悉线程中可用的变量、函数与类。在线程调用的 JavaScript 脚本文件中所有可用的变量、函数和类说明如下所示。

☑ self：self 关键字用来表示本线程范围内的作用域。

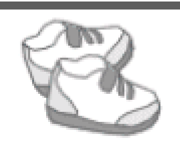
☑ postMessage(meseage)：向创建线程的源窗口发送消息。

☑ onmessage：获取接收消息的事件句柄。

☑ importScripts(urls)：导入其他 JavaScript 脚本文件。参数为该脚本文件的 URL 地址，可以导入多个脚本文件。导入的脚本文件必须与使用该线程文件的页面在同一个域中，并在同一个端口中。

☑ importScripts("worker.js","worker1.js","worker2.js");

☑ navigator 对象：与 window.navigator 对象类似，具有 appName、platform、userAgent、appVersion 属性。它们可以用来标识浏览器的字符。



- ☑ sessionStorage/localStorage：在线程中可以使用 Web Storage。
- ☑ XMLHttpRequest：在线程中可以处理 Ajax 请求。
- ☑ Web Workers：在线程中可以嵌套线程。
- ☑ setTimeout()/setInterval()：在线程中可以实现定时处理。
- ☑ close：结束本线程。
- ☑ eval()、isNaN()、escape()等：可以使用所有 JavaScript 核心函数。
- ☑ object：可以创建和使用本地对象。
- ☑ WebSockets：可以使用 Web Sockets API 向服务器发送和接收信息。



Note

视频讲解

8.1.3 Workers 通信

使用后台线程时不能访问页面或窗口对象，但是并不代表后台线程不能与页面之间进行数据交互。为了实现页面与 Web Workers 通信，可以调用 postMessage()函数传入所需数据。同时将建立一个监听器，用来监听由 Web Workers 发送到页面的消息。为建立页面和 Web Workers 之间的通信，首先在页面中添加对 postMessage()函数的调用，如下所示。

```
document.getElementById("helloButton").onclick = function() {
    worker.postMessage("你好");
}
```

当用户单击按钮后，相应信息会被发送给 Web Workers，然后将事件监听器添加到页面中，用来监听从 Web Workers 发来的信息。

```
worker.addEventListener("message", messageHandler, true);
function messageHandler(e) {
    //来自 worker 的处理信息
}
```

编写 HTML5 Web Workers JavaScript 文件。在该文件中，需要添加事件监听器以监听发来的消息，并且通过调用 postMessage()函数实现与页面之间的通信。

为了完成页面与 Web Worker 之间的通信功能，首先，添加代码调用 postMessage()函数。例如，在 messageHandler()函数中可以添加如下代码。

```
function messageHandler(e) {
    postMessage("worker 说： " + e.data + " too");
}
```

接下来，在 Web Workers JavaScript 文件中添加事件监听器，以处理从页面发来的信息：

```
addEventListener("message", messageHandler, true);
```

接收到信息后会马上调用 messageHandler()函数以保证信息能及时返回。

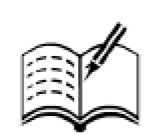
通过 postMessage()函数将对象传递到 workers 或者从中返回对象，这些对象将被自动转换为 JSON 格式。

```
var onmessage = function(e){
     postMessage(e.data);
 };
```

注意：在 workers 中进出的对象不能包含函数和循环引用，因为 JSON 不支持它们。



在 Web Workers 脚本中如果发生未处理的错误，会引发 Web Workers 对象的错误事件。特别是在调试用到 Web Workers 脚本时，对错误事件的监听就显得尤为重要。下面显示的是 Web Workers JavaScript 文件中的错误处理函数，它将错误记录在控制台上。



```
function errorHandler(e) {
    console.log(e.message, e);
}
```

为了处理错误，还必须在主页上添加一个事件监听器：

```
worker.addEventListener("error", errorHandler, true);
```

当 worker 发生运行时错误时，它的 onerror 事件就会被触发。该事件接收一个 error 的事件，该事件不会冒泡，并且可以取消。要取消该事件可以使用 preventDefault()方法。该错误事件有 3 个属性：
message：可读的错误信息。
filename：发生错误的脚本文件名称。
lineno：发生错误的脚本所在文件的行数。

Web Workers 不能自行终止，但能够被启用它们的页面所终止。调用 terminate()函数可以终止后台进程。被终止的 Web Workers 将不再响应任何信息或者执行任何其他的计算。终止之后，Worker 不能被重新启动，但可以使用同样的 URL 创建一个新的 Worker。

```
worker.terminate();
```

如果需要马上终止一个正在运行中的 worker，你可以调用它的 terminate()方法：

```
myWorker.terminate();
```

这样一个 worker 进程就结束了。

8.1.4　使用 Web Workers


视频讲解

【示例 1】下面示例演示了如何使用 Web Workers 在控制台显示一个提示信息。

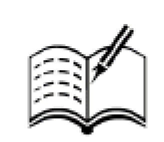
首先，设计主页面代码（index.html）。

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<script type="text/javascript">
    //Web 页主线程
    //创建一个 Worker 对象并向它传递将在新线程中执行的脚本的 URL
    var worker = new Worker("worker.js");
    worker.postMessage("hello world");//向 worker 发送数据
    worker.onmessage = function(evt) {//接收 worker 传过来的数据函数
        console.log(evt.data);//输出 worker 发送来的数据
    }
</script>
</head>
<body></body>
</html>
```



下面是线程脚本文件 worker.js 代码：

```
onmessage = function(evt) {
    var d = evt.data;//通过 evt.data 获得发送来的数据
    postMessage(d);//将获取到的数据发送回主线程
}
```



在 Chrome 浏览器中访问主页文件，则可以在控制台中看到输出的信息，表示程序执行成功，如图 8.1 所示。

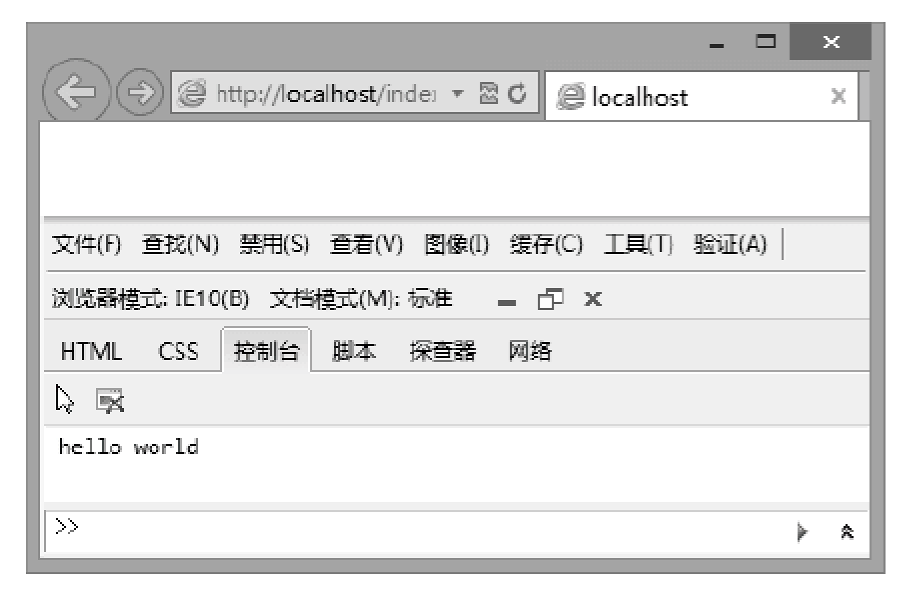


图 8.1　在控制台中查看信息

通过上面示例可以看到使用 Web Workers 应该包括下面两部分：

定义主页线程

- 通过 worker = new Worker(url)加载一个 JavaScript 文件，创建一个 Worker，同时返回一个 worker 实例。
- 通过 worker.postMessage(data)方法向 worker 发送数据。
- 绑定 worker.onmessage 事件接收 worker 响应的数据。
- 使用 worker.terminate()可以终止一个 worker 执行。

定义 Worker 线程

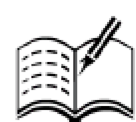
- 通过 postMessage(data)方法向主线程发送数据。
- 绑定 onmessage 事件接收主线程发送过来的数据。

【示例 2】下面示例演示如何创建 Web Workers，手动控制 Web Workers 与页面进行通信的一般方法，同时设置如何处理异常，以及如何停止 Worker 任务处理。

首先，设计主页文件（index.html），并在该文件脚本中定义一个主线程。

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
</head>
<p id="support">你的浏览器不支持 HTML5 Web Workers</p>
<button id="stopButton" >停止任务</button>
<button id="helloButton" >发送消息</button>
<script>
function stopWorker() { //终止线程
    worker.terminate();
}
function messageHandler(e) { //显示线程响应信息
```





```
        console.log(e.data);
    }
    function errorHandler(e) { //线程错误处理
        console.warn(e.message, e);
    }
    function loadDemo() {
        if( typeof (Worker) !== "undefined") {
            document.getElementById("support").innerHTML = "你的浏览器支持 HTML5 Web Workers";
            worker = new Worker("worker.js");
            worker.addEventListener("message", messageHandler, true);
            worker.addEventListener("error", errorHandler, true);
            document.getElementById("helloButton").onclick = function() {
                worker.postMessage("ok");
            }
            document.getElementById("stopButton").onclick = stopWorker;
        }
    }
    window.addEventListener("load", loadDemo, true);
    </script>
```

然后，设计线程脚本文件（worker.js）的代码：

```
function messageHandler(e) {
    postMessage("worker says: " + e.data + " too");
}
addEventListener("message", messageHandler, true);
```

在主页和线程文件中，分别使用 addEventListener 方法把回调函数绑定到线程监听事件中。

最后，在 Chrome 浏览器中访问主页文件，单击“发送消息”按钮，则可以在控制台中看到输出的信息，表示程序手动控制线程交互执行成功，如图 8.2 所示。

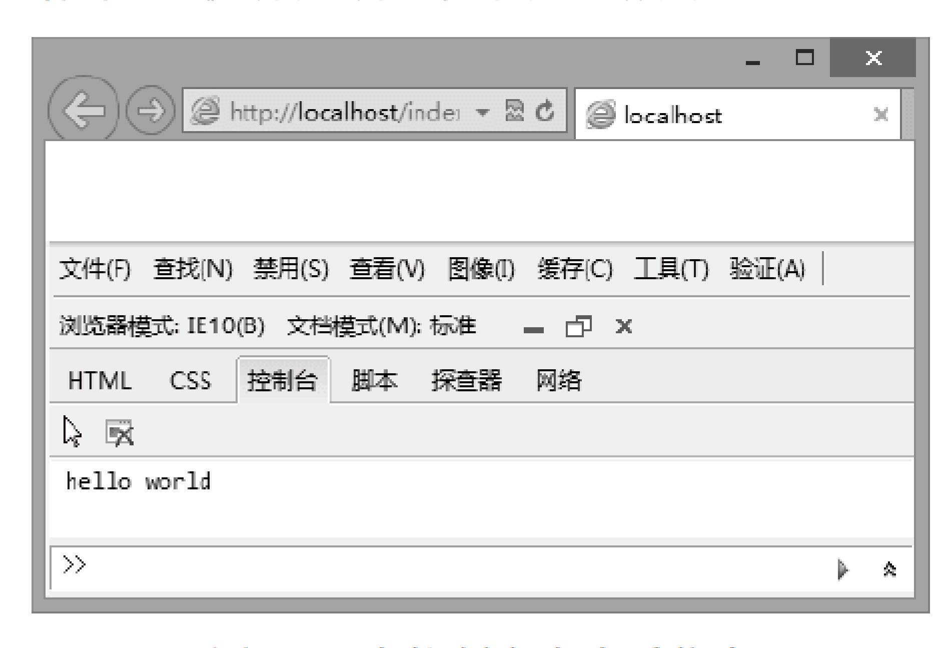


图 8.2　在控制台中查看信息

【示例 3】使用 addEventListener 方法注册后台线程的响应事件比较麻烦，当然，我们也可以把它修改为下面这种传统写法。

主线程脚本（index.html）

```
window.onload = function() {
    if( typeof (Worker) !== "undefined") {
        document.getElementById("support").innerHTML = "你的浏览器支持 HTML5 Web Workers";
        worker = new Worker("worker.js");
```




```
            worker.onmessage = function(e) {
                console.log(e.data);
            }
            worker.onerror = function(e) {
                console.warn(e.message, e);
            }
            document.getElementById("helloButton").onclick = function() {
                worker.postMessage("ok");
            }
            document.getElementById("stopButton").onclick = function() {
                worker.terminate();
            };
        }
    }
```

Worker 线程文件（worker.js）

```
    onmessage = function(e) {
        postMessage("worker says: " + e.data );
    }
```

8.2　案 例 实 战

本节将通过多个实例演示如何灵活应用 Web Workers，实现并发式应用程序开发。

视频讲解

8.2.1　求和运算

本示例设计一个文本框，允许用户在该文本框中输入数字，然后单击按钮，在后台计算从 1 到给定数值的和。虽然对于从 1 到给定数值的求和计算只需要用一个求和公式就可以了，但是本示例中为了展示后台线程的使用方法，采取循环计算的方法。

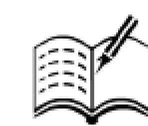
【示例 1】为了方便比较单线程与多线程的运算差异，首先采用传统方式设计一个单线程计算页面，主要代码如下。

```
<script type="text/javascript">
function calculate() {
    var num = parseInt(document.getElementById("num").value, 10);
    var result = 0;
    for (var i = 0; i <= num; i++) {
       result += i;
    }
    alert("合计值为" + result + "。");
}
</script>

输入数值:<input type="text" id="num">
<button onclick="calculate()">计算</button>
```

保存页面，然后在浏览器中预览，执行上面这段代码，在文本框中输入数值，然后单击“计算”





按钮。可以看到，在弹出提示对话框之前，用户是不能在该页面上执行操作的。虽然在文本框中输入比较小的值时，不会有什么延迟问题，但是当用户在该文本框中输入特别大的数字，浏览器运行时间明显延迟，如图 8.3 所示。

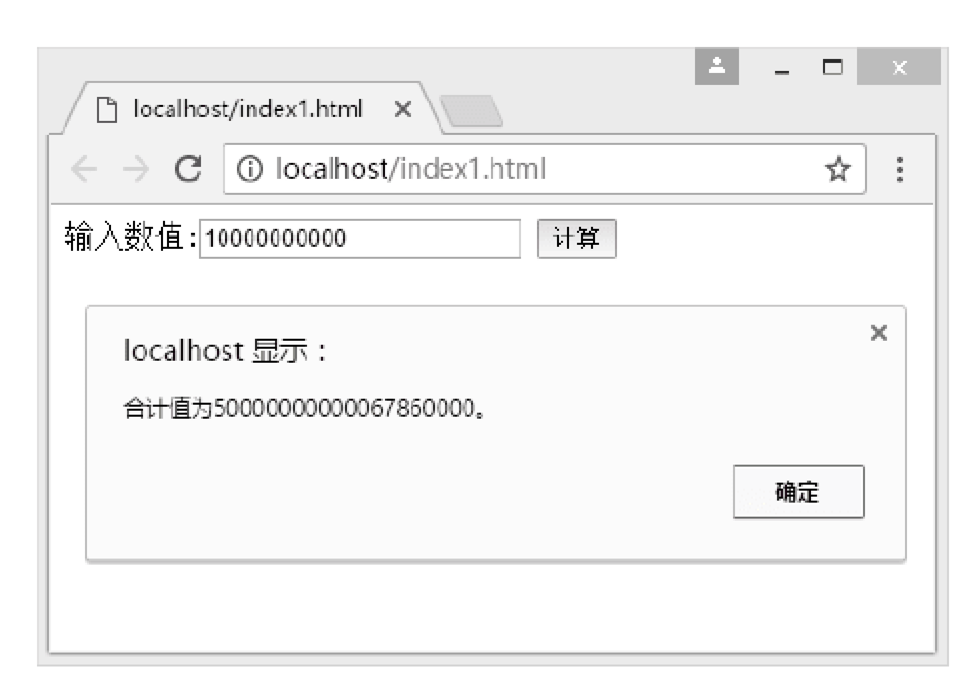


图 8.3　Safari 浏览器运行效果

【示例 2】重写该页面脚本，使用 Web Workers 把页面中比较耗时的运算放在后台运行，这样在上例的文本框中无论输入多么大的数值都可以正常运算了。

第 1 步，设计主页面，在该页面中创建一个 Worker，然后导入汇总计算的外部 JavaScript 文件。通过 postMessage()方法将用户输入的数字传递给 Worker，并通过 onmessage 事件回调函数接收运算的结果。

```
<script type="text/javascript">
var worker = new Worker("SumCalculate.js");//创建执行运算的线程
worker.onmessage = function(event) {//接收从线程中传出的计算结果
    alert("合计值为" + event.data + "。");
};
function calculate() {
    var num = parseInt(document.getElementById("num").value, 10);
    worker.postMessage(num); //将数值传给线程
}
</script>

输入数值:<input type="text" id="num">
<button onclick="calculate()">计算</button>
```

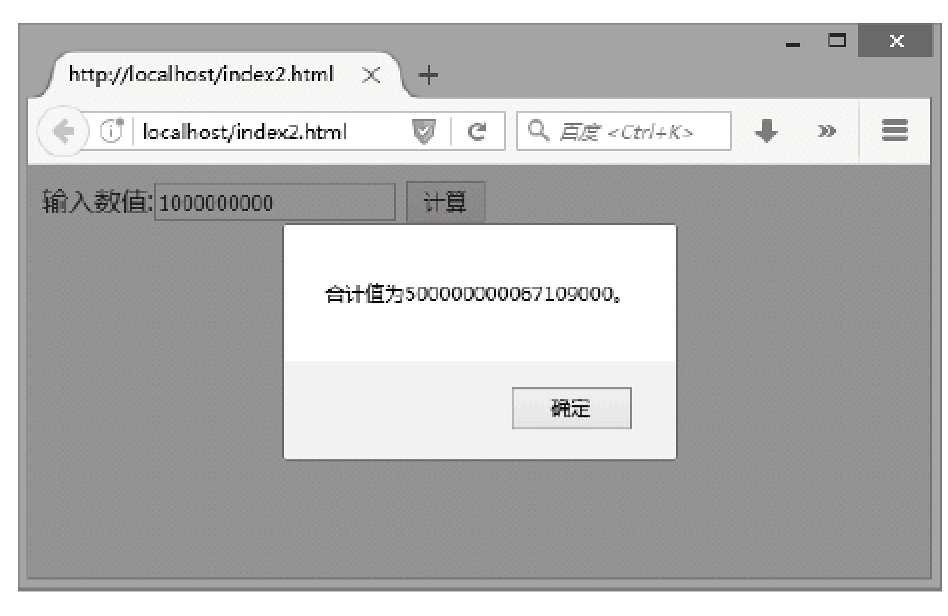
第 2 步，对于给定值的求和运算放到线程中单独执行，且把线程代码单独存储在 SumCalculate.js 脚本文件中。

```
onmessage = function(event) {
    var num = event.data;
    var result = 0;
    for (var i = 0; i <= num; i++)
        result += i;
    postMessage(result); //向线程创建源送回消息
}
```

第 3 步，在支持 Web Workers 的浏览器中预览，如 Firefox、Safari、Chrome、Opera 等浏览器，在 Firefox 中的运行结果如图 8.4 所示。







示例效果

图 8.4　Firefox 浏览器多线程运行效果



视频讲解

8.2.2　过滤运算

本示例设计在页面上随机生成一个整数的数组，然后将该整数数组传入线程，让后台帮助挑选出该数组中可以被 3 整除的数字，然后显示在页面表格中。读者可以借助这种设计思路，实现把字符串、数组、列表中的数据都采取该方法显示在页面表格、表单控件甚至统计图中。

【操作步骤】

第 1 步，设计前台页面代码，该页面的 HTML 代码部分包含一个空白表格，在前台脚本中随机生成整数数组，然后送到后台线程挑选出能够被 3 整除的数字，再传回前台脚本，在前台脚本中根据挑选结果动态创建表格中的行、列，并将挑选出来的数字显示在表格中。

```
<style type="text/css">
body { font: normal 11px auto "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif; color: #4f6b72; background: #E6EAE9; }
table { width: 700px; padding: 0; margin: 0; }
td { border-right: 1px solid #C1DAD7; border-bottom: 1px solid #C1DAD7; background: #fff; font-size:11px; padding: 6px 6px 6px 12px; color: #4f6b72; text-align:center; }
</style>
<script type="text/javascript">
var intArray=new Array(200);//随机数组
var intStr="";
//生成 200 个随机数
for(var i=0;i<200;i++){
    intArray[i]=parseInt(Math.random()*200);
    if(i!=0)
        intStr+=";"; //用分号作为随机数组的分隔符
    intStr+=intArray[i];
}
//向后台线程提交随机数组
var worker = new Worker("script.js");
worker.postMessage(intStr);
//从线程中取得计算结果
worker.onmessage = function(event) {
    if(event.data!="") {
        var j,k,tr,td;
        var intArray=event.data.split(";");
        var table=document.getElementById("table");
        for(var i=0;i<intArray.length;i++){
```




```
                j=parseInt(i/10,0);
                k=i%10;
                if(k==0) {//如果该行不存在，则添加行
                    tr=document.createElement("tr");
                    tr.id="tr"+j;
                    table.appendChild(tr);
                }
                else {//如果该行存在，则获取该行
                    tr=document.getElementById("tr"+j);
                }
                td=document.createElement("td");
                tr.appendChild(td);
                td.innerHTML=intArray[j*10+k];
            }
        }
    };
</script>

<table id="table"></table>
```

第 2 步，将后台线程中需要处理的任务代码存放在脚本文件 script.js 中，详细代码如下。

```
onmessage = function(event) {
    var data = event.data;
    var returnStr;
    var intArray=data.split(";");
    returnStr="";
    for(var i=0;i<intArray.length;i++){
        if(parseInt(intArray[i])%3==0) {
            if(returnStr!="")
                returnStr+=";";
            returnStr+=intArray[i];
        }
    }
    postMessage(returnStr); //返回 3 的倍数拼接成的字符串
}
```

第 3 步，在浏览器中预览，运行结果如图 8.5 所示。

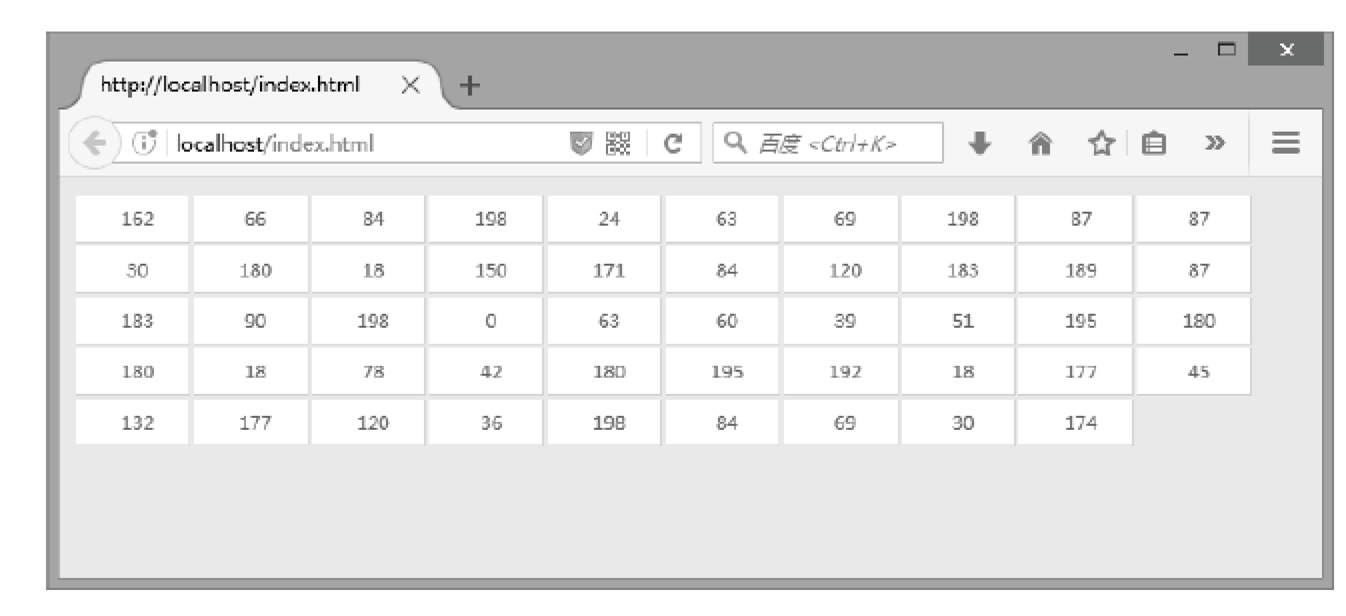


图 8.5　在后台过滤值



示 例 效 果



视 频 讲 解



Note



视 频 讲 解

8.2.3 并发运算



线 上 阅 读

本示例将在 8.2.2 节示例基础上，把主页脚本中随机生成数组的工作放到后台线程中，然后使用另一个子线程在随机数组中挑选可以被 3 整除的数字。对于数组的传递以及挑选结果的传递均采用 JSON 对象来进行转换，以验证是否能在线程之间进行 JavaScript 对象的传递工作。

具体操作步骤请扫码学习。

8.2.4 多运算通信

本示例继续在前面示例基础上，将创建随机数组的工作也放到了一个单独的子线程中，在该线程中创建随机数组，然后将随机数组传递到另一个子线程中进行能够被 3 整除的数字挑选工作，最后把挑选结果传递回主页面进行显示。

设计思路：

当主线程嵌套多个子线程时，子线程之间可以通过下面几个步骤进行通信。



线 上 阅 读

第 1 步，先创建发送数据的子线程。

第 2 步，执行子线程中的任务，然后把要传递的数据发送给主线程。

第 3 步，在主线程接收到子线程传回来的消息时，创建接收数据的子线程，然后把发送数据的子线程中返回的消息传递给接收数据的子线程。

第 4 步，执行接收数据子线程中的代码。

具体操作步骤请扫码学习。



视 频 讲 解

8.2.5 数列运算

由于 JavaScript 是单线程执行的，在求数列的过程中浏览器不能执行其他脚本，UI 渲染线程也会被挂起，从而导致浏览器进入“假死”状态。下面示例尝试使用 Web Workers 将数列计算过程放入一个新线程里，避免单线程计算所带来的问题。运行示例，在控制台中看到输出的信息，如图 8.6 所示。

线 上 阅 读

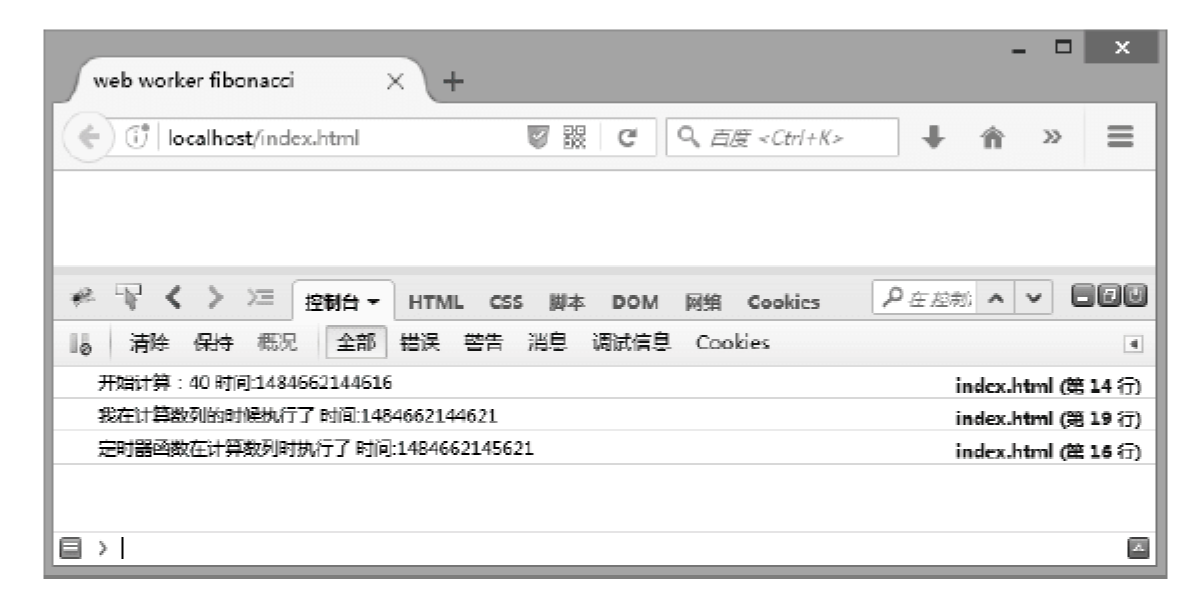


图 8.6 在控制台中查看信息

具体操作步骤请扫码学习。

8.3 在 线 练 习



在 线 练 习

使用 Web Worker 实现 Web 平台上的多线程处理功能。

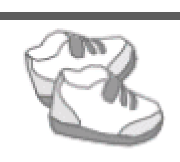
第 9 章

位置信息

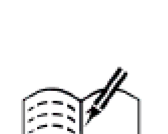
Geolocation API 是 HTML5 新增的地理位置应用程序接口，它提供了一个可以准确感知浏览器用户当前位置的方法。如果浏览器支持，且设备具有定位功能，就能够直接使用这组 API 来获取当前位置信息。该 Geolocation API 可以应用于移动设备中的地理定位，允许用户在 Web 应用程序中共享位置信息，使其能够享受位置感知服务。

【学习重点】

- 了解位置信息。
- 使用 Geolocation API。
- 根据位置信息设计简单的定位应用。



9.1 Geolocation API 基础



Note

在 HTML5 Geolocation API 之前，基于 IP 地址的地理定位方法是获得位置信息的唯一方式，但其返回的位置信息通常并不靠谱。基于 IP 地址的地理定位的实现方式是，自动查找用户的 IP 地址，然后检索其注册的物理地址。

9.1.1 Geolocation API 应用场景

应用场景 1：设计一个 Web 应用程序，向用户提供附近商店打折优惠信息。使用 HTML5 Geolocation API，可以请求用户共享他们的位置，如果他们同意，应用程序就可以向其提供相关信息，告诉用户去附近哪家商店可以挑选到打折的商品。

应用场景 2：构建计算行走（跑步）路程的应用程序。想象一下，在开始跑步时通过手机浏览器启动应用程序的记录功能。在用户移动过程中，应用程序会记录已跑过的距离，还可以把跑步过程对应的坐标显示在地图上，甚至可以显示出海拔信息。如果用户正在和其他选手一起参加跑步比赛，应用程序甚至可以显示其对手的位置。

应用场景 3：基于 GPS 导航的社交网络应用，可以用它看到好友们当前所处的位置，如知道了好友的方位，就可以挑选合适的咖啡馆。此外，还有很多特殊的应用。

9.1.2 位置信息来源

HTML5 Geolocation API 不指定设备使用哪种底层技术来定位应用程序的用户。相反，它只是用于检索信息的 API，而且通过该 API 检索到的数据只具有某种程度的准确性。它并不能保证设备返回的实际位置是精确的。设备可以使用下列数据。

- ☑ IP 地址。
- ☑ 三维坐标。
  - ➢ GPS 全球定位系统。
  - ➢ 从 RFID、Wi-Fi 和蓝牙到 Wi-Fi 的 MAC 地址。
  - ➢ GSM 或 CDMA 手机的 ID。
- ☑ 用户自定义数据。

为了保证更高的准确度，许多设备使用一个或多个数据源的组合。

9.1.3 位置信息表示方式

位置信息主要由一对纬度和经度坐标组成，例如：

```
Latitude: 39.17222, Longitude: -120.13778
```

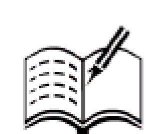
在这里，纬度（距离赤道以北或以南的数值表示）是 39.17222，经度（距离英国格林尼治以东或以西的数值表示）是 120.13778，经纬度坐标可以用以下两种方式表示：

- ☑ 十进制格式，如 39.17222。
- ☑ DMS 角度格式，如 39°20'。

HTML5 Geolocation API 返回坐标的格式为十进制格式。



除了纬度和经度坐标，HTML5 Geolocation 还提供位置坐标的准确度，并提供其他一些元数据，具体情况取决于浏览器所在的硬件设备，这些元数据包括海拔、海拔准确度、行驶方向和速度等。如果这些元数据不存在，则返回 null。



9.1.4　获取位置信息

HTML5 Geolocation API 的使用方法相当简单。请求一个位置信息，如果用户同意，浏览器就会返回位置信息，该位置信息是通过支持 HTML5 地理定位功能的底层设备。例如，笔记本电脑或手机提供给浏览器的位置信息由纬度、经度坐标和一些其他元数据组成。有了这些位置信息就可以构建引人注目的位置感知类应用程序。

在 HTML5 中，为 window.navigator 对象新增了一个 geolocation 属性，可以使用 Geolocation API 访问该属性，window.navigator 对象的 geolocation 属性包含三种方法，利用这些方法可以实现位置信息的读取。

使用 getCurrentPosition 方法可以取得用户当前的地理位置信息，该方法的用法如下所示。

```
void getCurrentPosition(onSuccess, onError, options) ;
```

第一个参数为获取当前地理位置信息成功时所执行的回调函数，第二个参数为获取当前地理位置信息失败时所执行的回调函数，第三个参数为一些可选属性的列表。其中，第二、三个参数为可选属性。

getCurrentPosition 方法中的第一个参数为获取当前地理位置信息成功时所执行的回调函数。该参数的使用方法如下所示。

```
navigator.geolocation.getCurrentPosition(function(position){
    //获取成功时的处理
}
```

在获取地理位置信息成功时执行的回调函数中用到了一个参数 position，它代表一个 position 对象。

getCurrentPosition 方法中的第二个参数为获取当前地理位置信息失败时所执行的回调函数。如果获取地理位置信息失败，可以通过该回调函数把错误信息提示给用户。当在浏览器中打开使用了 Geolocation API 来获得用户当前位置信息的页面时，浏览器会询问用户是否共享位置信息。如果在该画面中拒绝共享的话，也会引起错误的发生。

该回调函数使用一个 error 对象作为参数，该对象具有以下两个属性：

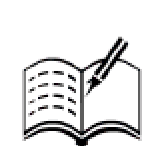
☑ code 属性：code 属性包含三个值，简单说明如下：
  - 当属性值为 1 时，表示用户拒绝了位置服务。
  - 当属性值为 2 时，表示获取不到位置信息。
  - 当属性值为 3 时，表示获取信息超时错误。

☑ message 属性：为一个字符串，在该字符串中包含了错误信息，这个错误信息在开发和调试时将很有用。因为有些浏览器中不支持 message 属性，如 Firefox。

在 getCurrentPosition 方法中使用第二个参数捕获错误信息的具体使用方法如下所示。

```
navigator.geolocation.getCurrentPosition(
        function(position){
            var cords = position.coords;
            showMap(coords.latitude, coords.longitude,coords.accuracy);
```





Note

```
        },
        //捕获错误信息
        function (error){
            var errorTypes = {
                1:位置服务被拒绝
                2:获取不到位置信息
                3:获取信息超时
             }
            alert( errorTypes[error.code]+ ":,不能确定当前地理位置");
        }
);
```

getCurrentPosition 方法中的第三个参数可以省略，它是一些可选属性的列表，这些可选属性说明如下。

☑ enableHighAccuracy

是否要求高精度的地理位置信息，这个参数在很多设备上设置了都没用，因为使用在设备上时需要结合设备电量、具体地理情况来综合考虑。因此，多数情况下把该属性设为默认，由设备自身来调整。

☑ timeout

对地理位置信息的获取操作做一个超时限制（单位为毫秒）。如果在该时间内未获取到地理位置信息，则返回错误。

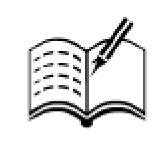
☑ maximumAge

对地理位置信息进行缓存的有效时间的单位为毫秒。例如，maximumAge：120000（1 分钟是 60000），如果 10 点整的时候获取过一次地理位置信息，10:01 的时候，再次调用 navigator.geolocation.getCurrentPosition 重新获取地理位置信息，则返回的依然为 10:00 时的数据（因为设置的缓存有效时间为 2 分钟）。超过这个时间后缓存的地理位置信息被废弃，尝试重新获取地理位置信息。如果该值被指定为 0，则无条件重新获取新的地理位置信息。

对于这些可选属性的具体设置方法如下所示。

```
navigator.geolocation.getCurrentPoeition(
        function(position){
          //获取地理位置信息成功时所做处理
        },
        function(error){
          //获取地理位置信息失败时所做处理
        },
        //以下为可选属性
        {
          //设缓存有效时间为 2 分钟
          maximumAge: 60*1000*2
          //5 秒钟内未获取到地理位置信息则返回错误
          timeout: 5000
        }
}
```





9.1.5 浏览器兼容性

各浏览器对 HTML5 Geolocation 的支持程度不同，并且还在不断更新。在 HTML5 的所有功能中，HTML5 Geolocation 是第一批被全部接受和实现的功能之一，这对于开发人员来说是个好消息。相关规范已达到一个非常成熟的阶段，不大可能做大的改变。各浏览器对 HTML5 Geolocation 的支持情况如表 9.1 所示。

表 9.1 浏览器支持概述

浏 览 器	说 明
IE	通过 Gears 插件支持
Firefox	3.5 及以上的版本支持
Opera	10 及以上的版本支持
Chrome	2.0 及以上的版本支持
Safari	4.0 及以上的版本支持

由于浏览器对它的支持程度不同，在使用之前最好先检查浏览器是否支持 HTML5 Geolocation API，确保浏览器支持其所要完成的所有工作。这样当浏览器不支持时，就可以提供一些替代文本，以提示用户升级浏览器或安装插件来增强现有浏览器功能。

```
function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("support").innerHTML = "支持 HTML5 Geolocation";
    } else {
        document.getElementById("support").innerHTML = "当前浏览器不支持 HTML5 Geolocation";
    }
}
```

在上面代码中，loadDemo()函数测试了浏览器的支持情况，这个函数是在页面加载的时候被调用的。如果存在地理定位对象，navigator.geolocation 调用将返回该对象，否则将触发错误。页面上预先定义的 support 元素会根据检测结果显示支持情况的提示信息。

9.1.6 监测位置信息

使用 watchCurrentPosition 方法可以持续获取用户的当前地理位置信息，它会定期地自动获取。watchCurrentPosition 方法的基本语法如下所示。

```
int watchCurrentPosition(onSuccess, onError, options) ;
```

该方法参数的说明与使用与 getCurrentPosition 方法相同。调用该方法后会返回一个数字，这个数字的用法与 JavaScript 脚本中 setInterval 方法的返回值用法类似，可以被 clearWatch 方法使用，以停止对当前地理位置信息的监视。

9.1.7 停止获取位置信息

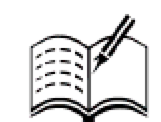
使用 clearWatch 方法可以停止对当前用户的地理位置信息的监视。具体用法如下所示。



```
void clearWatch(watchld);
```

参数 watchId 为调用 watchCurrentPosition 方法监视地理位置信息时的返回参数。

9.1.8 保护隐私



HTML5 Geolocation 规范提供了一套保护用户隐私的机制。除非得到用户明确许可，否则不可获取位置信息。

【操作步骤】

第 1 步，用户从浏览器中打开位置感知应用程序。

第 2 步，应用程序 Web 页面加载，然后通过 Geolocation 函数调用请求位置坐标。浏览器拦截这一请求，然后请求用户授权。

第 3 步，如果浏览器从其宿主设备中检索坐标信息，如 IP 地址、Wi-Fi 或 GPS 坐标，这是浏览器的内部功能。

第 4 步，浏览器将坐标发送给受信任的外部定位服务，它返回一个详细位置信息，并将该位置信息发回给 HTML5 Geolocation 应用程序。

提示： 应用程序不能直接访问设备，它只能请求浏览器来代表它访问设备。

访问使用 HTML5 Geolocation API 的页面时，会触发隐私保护机制。如果仅仅是添加 HTML5 Geolocation 代码，而不被任何方法调用，则不会触发隐私保护机制。只要所添加的 HTML5 Geolocation 代码被执行，浏览器就会提示用户应用程序要共享位置。执行 HTML5 Geolocation 的方式很多。例如，调用 navigator.geolocation.getCurrentPosition 方法等。

除了询问用户是否允许共享其位置之外，Firefox 等一些浏览器还可以让用户选择记住该网站的位置服务权限，以便下次访问的时候不再弹出提示框，类似在浏览器中记住某些网站的密码。

9.1.9 处理位置信息

因为位置数据属于敏感信息，所以接收到之后，必须小心地处理、存储和重传。如果用户没有授权存储这些数据，那么应用程序应该在相应任务完成后立即删除。如果要重传位置数据，建议先对其进行加密。在收集地理定位数据时，应用程序应该着重提示用户以下内容：

- ☑ 会收集位置数据。
- ☑ 为什么收集位置数据。
- ☑ 位置数据将保存多久。
- ☑ 怎样保证数据的安全。
- ☑ 如果用户同意共享，位置数据怎样共享。
- ☑ 用户怎样检查和更新他们的位置数据。

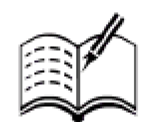
9.1.10 使用 position

如果获取地理位置信息成功，则可以在获取成功后的回调函数中通过访问 position 对象的属性来得到这些地理位置信息。position 对象具有如下属性。

- ☑ latitude：当前地理位置的纬度。
- ☑ longitude：当前地理位置的经度。



- ☑ altitude：当前地理位置的海拔（不能获取时为 null）。
- ☑ accuracy：获取到的纬度或经度的精度（以米为单位）。
- ☑ altitudeAccurancy：获取到的海拔的精度（以米为单位）。
- ☑ heading：设备的前进方向。用面朝正北方向的顺时针旋转角度来表示（不能获取时为 null）。
- ☑ speed：设备的前进速度（以米/秒为单位，不能获取时为 null）。
- ☑ timestamp：获取地理位置信息时的时间。



【示例】下面示例使用 getCurrentPosition 方法获取当前位置的地理信息，并且在页面中显示 position 对象中的所有属性。

```
<script type="text/javascript" src=http://maps.google.com/maps/api/js?sensor=false></script>
<script type="text/javascript">
function showObject(obj,k){   //递归显示 object
    if(!obj){return;}
    for(var i in obj){
        if(typeof(obj[i])!="object" || obj[i]==null){
            for(var j=0;j<k;j++){
                document.write("    ");
            }
            document.write(i + " : " + obj[i] + "<br/>");
        } else {
            document.write(i + " : " + "<br/>");
            showObject(obj[i],k+1);
        }
    }
}
function get_location(){
    if(navigator.geolocation)
navigator.geolocation.getCurrentPosition(show_map,handle_error,{enableHighAccuracy:true,maximumAge:1000});
    else    alert("你的浏览器不支持使用 HTML5 来获取地理位置信息。");
}
function handle_error(err){ //错误处理
     switch(err.code){
         case 1 :
             alert("位置服务被拒绝。");
             break;
         case 2:
             alert("暂时获取不到位置信息。");
             break;
         case 3:
             alert("获取信息超时。");
             break;
         default:
             alert("未知错误。");
             break;
    }
}
function show_map(position){   //显示地理信息
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
```

```
        showObject(position,0);
    }
    get_location();
    </script>
    <div id="map" style="width:400px; height:400px"></div>
```




这段代码运行结果在不同设备的浏览器上也不同，具体运行结果取决于运行浏览器的设备。



视 频 讲 解

示 例 效 果

9.2　案例：设计位置地图

本示例介绍如何在页面上显示一幅 Google 地图，并且把用户的当前地理位置标注在地图上面，如果用户的位置发生改变，将把之前在地图上的标记自动更新到新的位置上。当然，要在页面中使用 Google 地图，需要使用到 Google Map API。

【操作步骤】

第 1 步，在页面中导入 Google Map API 的脚本文件，导入方法如下所示。

```
<script type="text/javascript" src=http://maps.google.com/maps/api/js?sensor=false />
```

第 2 步，设定地图参数，设定方法如下所示。

```
//指定 Google 地图上的一个坐标点，同时指定该坐标点的横坐标和纵坐标
var latlng = new google.maps.LatLng(coords.latitude, coords.longitude);
var myOptions = {
    zoom: 14, //设定放大倍数
    center: latlng, //将地圈中心点设定为指定的坐标点
    mapTypeId: google.maps.MapTypeId.ROADMAP //指定地圈类型
};
```

本例将用户当前位置的纬度、经度设定为页面打开时 Google 地图的中心点。

第 3 步，创建地图，并在页面中显示。

```
var map1= new google.maps.Map(document.getElementById("map"), myOptions);
```

上面代码将地图显示在 id 为 map 的 div 元素中。

第 4 步，在地图上创建标记。

```
var marker = new google.maps.Marker({
    position: latlng, //将前面指定的坐标点标注出来
    map: map1 //设置在 map1 变量代表的地图中标注
});
```

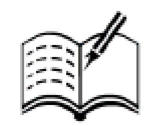
第 5 步，设置标注窗口并指定标注窗口中注释文字。

```
var infowindow = new google.maps.InfoWindow({
    content: "当前位置!"    //指定标注窗口中注释文字
});
```

第 6 步，打开标注窗口。

```
infowindow.open(map1, marker);
```





示例主要代码如下所示。

```
<script type="text/javascript" src=http://maps.google.com/maps/api/js?sensor=false></script>
<script type="text/javascript">
  function init() {
    //取得当前地理位置
    navigator.geolocation.getCurrentPosition(function(position) {
        var coords = position.coords;
        //设定地图参数，将用户的当前位置的纬度、经度设定为地图的中心点
        var latlng = new google.maps.LatLng(coords.latitude, coords.longitude);
        var myOptions = {
          zoom: 14,
          center: latlng,
          mapTypeId: google.maps.MapTypeId.ROADMAP
        };
        //创建地图并在“map”div 中显示
        var map1;
        map1= new google.maps.Map(document.getElementById("map"), myOptions);
        //在地图上创建标记
        var marker = new google.maps.Marker({
            position: latlng,
            map: map1
        });
        //设定标注窗口，并指定该窗口中的注释文字
        var infowindow = new google.maps.InfoWindow({
          content: "当前位置!"
        });
        //打开标注窗口
        infowindow.open(map1, marker);
    });
  }
</script>
<body onload="init()">
  <div id="map" style="width:400px; height:400px"></div>
</body>
```

9.3　在线练习

使用 HTML5 Geolocation API 来获取当前位置信息。



在线练习

第10章 历史记录

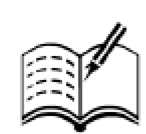
在HTML5之前，使用JavaScript实现历史记录的更新，都会触发一个页面刷新，这个更新过程将耗费大量时间和资源。在很多情况下，这种刷新是没有必要的，它会导致页面内容重复加载。HTML5的History API允许在不刷新页面的前提下，通过JavaScript方式更新页面内容。

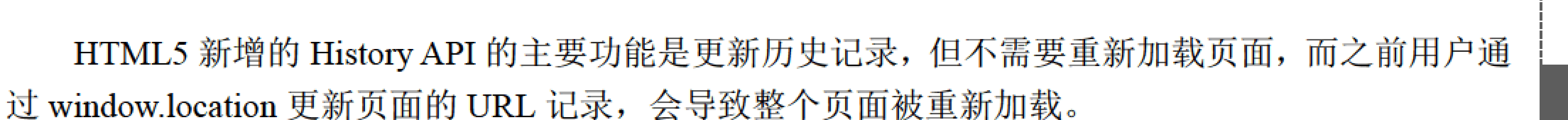
【学习重点】

- 正确使用History API。
- 使用Ajax+History API设计无刷新页面。



10.1　History API 基础





HTML5 新增的 History API 的主要功能是更新历史记录，但不需要重新加载页面，而之前用户通过 window.location 更新页面的 URL 记录，会导致整个页面被重新加载。

10.1.1　了解 History API

History API 新增如下历史记录的控制功能：

☑　允许用户在浏览器历史记录中添加项目。

☑　在不刷新页面的前提下，允许显式改变浏览器地址栏中的 URL 地址。

☑　新添了一个当激活的历史记录发生改变时触发的事件，如前进或后退浏览页面。

通过这些新增功能和事件，可以实现在不刷新页面的前提下，动态改变浏览器地址栏中的 URL 地址，动态修改页面所显示的资源。

History API 执行过程如下。

第 1 步，通过 Ajax 向服务器端请求页面需要更新的信息。

第 2 步，使用 JavaScript 加载并显示更新的页面信息。

第 3 步，通过 History API 在不刷新页面的前提下，更新浏览器地址栏中的 URL 地址。

在整个处理过程中，页面信息得到更新，浏览器的地址栏也发生了变化，但是页面并没有被刷新。实际上，History API 的诞生，主要任务就是为了解决 Ajax 技术与浏览器历史记录之间存在的冲突。

提示： 完善 Ajax 与 History API 融合，需要注意两个问题：

☑　将 Ajax 请求的地址嵌入<a>标记的 href 属性中。

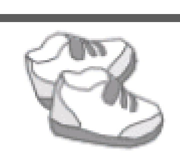
☑　确保在 JavaScript 的 click 事件处理程序中返回 true，这样当用户使用中键单击或命令单击时不会导致程序被意外覆盖。

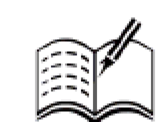
目前，IE 10+、Firefox 4+、Chrome 8+、Safari 5+、Opera 11+等主流版本浏览器支持 HTML5 中的 History API。

提示： 如果只修改 URL 中的 hash（历史记录），则不会导致页面被刷新。使用传统的 hashbang 方法可以改变页面的 URL，但不刷新页面。Twitter 网站就使用这种方法，不过这种方法广受诟病，毕竟 hash 在 location 中并不被作为一个真正的资源来对待。2012 年 Twitter 抛弃了 hashbang 方法，推出 pushstate()方法，随后各浏览器支持了这个规范。

如果想大范围地使用 History API 技术，可以考虑使用一些专有的工具，如 pjax（https://github.com/defunkt/jquery-pjax），它是一个 jQuery 插件，使用它可以大大提高用户同时使用 Ajax 和 pushState()方法进行开发的速度，不过它只支持那些使用 History API 接口的现代浏览器。

注意： 对于不支持 History API 接口的浏览器，可以使用 history.js 进行兼容，它使用旧的 URL hash 的方式来实现同样的功能。下载地址为 https://github.com/browserstate/history.js/。





Note

10.1.2 使用 History API

window 通过 History 对象提供对浏览器历史记录的访问能力，允许用户在历史记录中自由前进和后退，而在 HTML5 中，还可以操纵历史记录中的数据。

☑ 在历史记录中后退

实现方法如下：

```
window.history.back();
```

这行代码等效于在浏览器的工具栏上单击“返回”按钮。

☑ 在历史记录中前进

实现方法如下：

```
window.history.forward();
```

这行代码等效于在浏览器中单击“前进”按钮。

☑ 移动到指定的历史记录点

可以使用 go()方法从当前会话的历史记录中加载页面。当前页面位置索引值为 0，上一页就是-1，下一页为 1，依此类推。

```
window.history.go(-1);        //相当于调用 back()
window.history.go(1);         //相当于调用 forward()
```

☑ length 属性

使用 length 属性可以了解历史记录栈中一共有多少页：

```
var numberOfEntries = window.history.length;
```

☑ 添加和修改历史记录条目

HTML5 新增 history.pushState()和 history.replaceState()方法，允许用户逐条添加和修改历史记录条目。使用 history.pushState()方法可以改变 referrer 的值，而在调用该方法后创建的 XMLHttpRequest 对象会在 HTTP 请求头中使用这个值。referrer 的值则是创建 XMLHttpRequest 对象时所处的窗口的 URL。

【示例】假设 http://mysite.com/foo.html 页面将执行下面 JavaScript 代码：

```
var stateObj = { foo: "bar" };
history.pushState(stateObj, "page 2", "bar.html");
```

这时浏览器的地址栏将显示 http://mysite.com/bar.html，但不会加载 bar.html 页面，也不会检查 bar.html 是否存在。

如果现在用户导航到 http://mysite.com/页面，然后单击后退按钮，此时地址栏将会显示 http://mysite.com/bar.html，并且页面会触发 popstate 事件，该事件状态对象会包含 stateObj 的一个副本。

如果再次单击后退按钮，URL 将返回 http://mysite.com/foo.html，文档将触发另一个 popstate 事件，这次的状态对象为 null，回退同样不会改变文档内容。

☑ pushState()方法

pushState()方法包含三个参数，简单说明如下：

第 1 个参数：状态对象。

状态对象是一个 JavaScript 对象直接量，与调用 pushState()方法创建的新历史记录条目相关联。



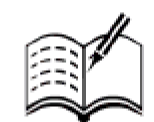
无论何时用户导航到新创建的状态，popstate 事件都会被触发，并且事件对象的 state 属性都包含历史记录条目的状态对象的副本。

第 2 个参数：标题。可以传入一个简短的标题，标明将要进入的状态。

Firefox 浏览器目前忽略该参数，考虑到未来可能会对该方法进行修改，传一个空字符串会比较安全。

第 3 个参数：可选参数，新的历史记录条目的地址。

浏览器不会在调用 pushState()方法后加载该地址，不指定的话则为文档当前 URL。



Note

提示：调用 pushState()方法，类似于设置 window.location='#foo'，它们都会在当前文档内创建和激活新的历史记录条目。但 pushState()有自己的优势:

- ☑ 新的 URL 可以是任意的同源 URL，与此相反，使用 window.location 方法时，只有仅修改 hash 才能保证停留在相同的 document 中。
- ☑ 根据个人需要决定是否修改 URL。相反，设置 window.location='#foo'，只有在当前 hash 值不是 foo 时才创建一条新历史记录。
- ☑ 可以在新的历史记录条目中添加抽象数据。如果使用基于 hash 的方法，只能把相关数据转码成一个很短的字符串。

注意：pushState()方法永远不会触发 hashchange 事件。

☑ replaceState()方法

history.replaceState()与 history.pushState()用法相同，都包含 3 个相同的参数。

不同之处：

pushState()是在 history 栈中添加一个新的条目，replaceState()是替换当前的记录值。例如，history 栈中有两个栈块，一个标记为 1，另一个标记为 2，现在有第三个栈块，标记为 3。当执行 pushState()时，栈块 3 将被添加到栈中，栈就有 3 个栈块了。而当执行 replaceState()时，将使用栈块 3 替换当前激活的栈块 2，history 的记录条数不变。

提示：为了响应用户的某些操作，需要更新当前历史记录条目的状态对象或 URL 时，使用 replaceState()方法会特别合适。

☑ popstate 事件

每当激活的历史记录发生变化时，都会触发 popstate 事件。如果被激活的历史记录条目是由 pushState()创建，或者是被 replaceState()方法替换的，popstate 事件的状态属性将包含历史记录的状态对象的一个副本。

注意：当浏览会话历史记录时，不管是单击浏览器工具栏中“前进”或者“后退”按钮，还是使用 JavaScript 的 history.go()和 history.back()方法，popstate 事件都会被触发。

☑ 读取历史状态

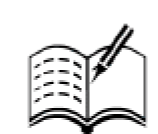
在页面加载时，可能会包含一个非空的状态对象。这种情况是会发生的，例如，如果页面中使用 pushState()或 replaceState()方法设置了一个状态对象，然后重启浏览器。当页面重新加载时，页面会触发 onload 事件，但不会触发 popstate 事件。但是，如果读取 history.state 属性，会得到一个与 popstate 事件触发时一样的状态对象。

可以直接读取当前历史记录条目的状态，而不需要等待 popstate 事件：

```
var currentState = history.state;
```



10.1.3　注意事项



Note

一般 History API 与 Ajax 结合使用才有价值，应用中主要掌握三个技术要点：第一，使用 Ajax 实现网页内容的更新；第二，使用 History API 实现浏览器历史记录的更新；第三，使用 History API 实时跟踪浏览器的导航响应，实现当浏览器的历史记录发生变化时，页面内容也应随之更新。

注意： 测试本章示例，用户需要搭建一个 Web 服务器，以 http://host/的形式去访问才能生效。如果在本地测试，以 file://的方式在浏览器打开，就会出现如下的问题：

Uncaught SecurityError: A history state object with URL 'file:///C:/xxx/xxx/xxx/xxx.html' cannot be created in a document with origin 'null'.

因为使用 pushState()方法修改的 URL 与当前页面的 URL 必须是同源的，而 file://形式打开的页面是没有 origin 的，所以会报该错误。

10.2　案 例 实 战

视频讲解

下面结合几个案例学习 History API 的具体应用。

10.2.1　设计导航页面

本例设计一个无刷新页面导航，在首页（index.html）包含一个导航列表，当用户单击不同的列表项目时，首页（index.html）的内容器（<div id="content">）会自动更新内容，正确显示对应目标页面的 HTML 内容，同时浏览器地址栏正确显示目标页面的 URL，但是首页并没有被刷新，而不是仅显示目标页面。演示效果如图 10.1 所示。



（a）显示 index.html 页面



（b）显示 news.html 页面

图 10.1　应用 History API

在浏览器工具栏中单击“后退”按钮，浏览器能够正确显示上一次单击的链接地址，虽然页面并没有被刷新，同时地址栏中正确显示上一次浏览页面的 URL，如图 10.2 所示。如果没有 History API 支持，使用 Ajax 实现异步请求时，工具栏中的“后退”按钮是无效的。

如果在工具栏中单击“刷新”按钮，则页面将根据地址栏的 URL 信息重新刷新页面，将显示独立的目标页面，效果如图 10.3 所示。

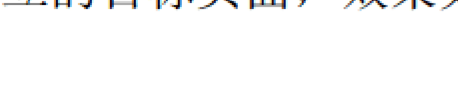








图 10.2　正确后退和前进历史记录



图 10.3　重新刷新页面显示效果

如果单击工具栏中的“后退”和“前进”按钮，会发现导航功能失效，页面总是显示目标页面，如图 10.4 所示。这说明使用 History API 控制导航与浏览器导航功能存在差异，一个是 JavaScript 脚本控制，一个是系统自动控制。





示 例 效 果

图 10.4　刷新页面之后工具栏导航失效

【操作步骤】

第 1 步，设计首页（index.html）。新建文档，保存为 index.html，构建 HTML 导航结构。

```
<h1>History API 示例</h1>
<ul id="menu">
    <li><a href="news.html">News</a></li>
    <li><a href="about.html">About</a></li>
    <li><a href="contact.html">Contact</a></li>
</ul>
```

```
<div id="content">
    <h2>当前内容页：index.html</h2>
</div>
```

第 2 步，由于本例使用 jQuery 框架，因此在文档头部位置导入 jQuery 库文件。

```
<script src="jquery/jquery-1.11.0.js" type="text/javascript"></script>
```

第 3 步，定义异步请求函数。该函数根据参数 url 值，异步加载目标地址的页面内容，并把它置入首页内容容器（<div id="content">）中，同时根据第 2 个参数 addEntry 的值执行额外操作。如果第 2 个参数值为 true，使用 history.pushState()方法把目标地址推入历史记录堆栈中。

```
function getContent(url, addEntry) {
    $.get(url)                                  //异步请求
    .done(function( data ) {
        $('#content').html(data);               //动态加载目标页面
        if(addEntry == true) {
            history.pushState(null, null, url);//把目标地址推入浏览器历史记录堆栈中
        }
    });
}
```

第 4 步，在页面初始化事件处理函数中，为每个导航链接绑定 click 事件，在 click 事件处理函数中调用 getContent()函数，同时阻止页面的刷新操作。

```
$(function(){
    $('#menu a').on('click', function(e){
        e.preventDefault();                     //阻止页面刷新操作
        var href = $(this).attr('href');
        getContent(href, true);                 //执行页面内容更新操作
        $('#menu a').removeClass('active');
        $(this).addClass('active');
    });
});
```

第 5 步，注册 popstate 事件，跟踪浏览器历史记录的变化，如果发生变化，则调用 getContent()函数更新页面内容，但是不再把目标地址添加到历史记录堆栈中。

```
window.addEventListener("popstate", function(e) {
    getContent(location.pathname, false);
});
```

第 6 步，设计其他页面。

☑ about.html

```
<h2>当前内容页：about.html</h2>
```

☑ contact.html

```
<h2>当前内容页：contact.html</h2>
```

☑ news.html

```
<h2>当前内容页：news.html</h2>
```





Note

10.2.2　设计无刷新网站

本例设计一个简单的网站，当用户选择一幅图片时，在下方将显示该技术对应的文字描述，同时高亮显示该图片，提示被选中状态。当在浏览器工具栏中单击“后退”按钮时，页面应该切换到上一个被选中的图片状态，同时图片下方的文字也要一并切换；当单击“前进”按钮时执行类似的响应操作，演示效果如图 10.5 所示。



（a）网站首页默认效果



（b）显示火药技术视图效果

图 10.5　设计主题宣传网站

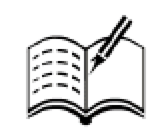
这样当单击一幅图片，然后将被更改的 URL 分享出去，共享用户可以通过这个 URL 访问对应的网页。这会带来一些更好的用户体验，并保证了 URL 和页面内容的一致性，从而减少了 Ajax 传统应用中 URL 与显示内容不一致的问题，这对于依赖 URL 的应用来说是一个障碍，会因此带给用户的一些困惑。

【操作步骤】

第 1 步，新建网站首页（index.html）结构。本示例的 HTML 代码非常简单：<div class="gallery">中包含了所有的链接，每个链接里有一幅图片，在下面放置一个空的<div class="content">容器，用来存放当图片被单击时显示图片介绍文字。

```
<div class="page-wrap">
        <div class="gallery">
            <a href="/zhaozhishu.php">
                <img src="images/zhaozhishu.png" alt="造纸术" class="zhaozhishu" data-name="zhaozhishu"/> </a>
            <a href="/huoyao.php">
                <img src="images/huoyao.png" alt="火药" class="huoyao" data-name="huoyao"/></a>
            <a href="/yinshuashu.php">
              <img src="images/yinshuashu.png" alt="印刷术" class="yinshuashu" data-name="yinshuashu"/> </a>
            <a href="/zhinanzhen.php">
              <img src="images/zhinanzhen.png" alt="指南针" class="zhinanzhen" data-name="zhinanzhen"></a>
        </div>
    <p class="selected">中国四大发明</p>
    <p class="highlight"></p>
    <div class="content"></div>
</div>
```





Note

> 提示：在设计结构时，要考虑页面的可访问性和优雅降级：如果没有 JavaScript，该页面仍然可以正常工作，单击图片可以跳转到对应的页面，然后单击“后退”按钮也可以回到之前的页面，效果如图 10.6 所示。



示例效果

图 10.6　无 JavaScript 状态下显示火药技术页面效果

第 2 步，新建 JavaScript 文件，保存为 images/app.js，然后在页面中导入该脚本文件。

```
<script src="images/app.js"></script>
```

第 3 步，在脚本文件中添加 JavaScript 代码。为<div class="gallery">容器中的每一个<a.>添加一个 click 事件处理程序。

```
var container = document.querySelector('.gallery');
container.addEventListener('click', function(e) {
    if (e.target != e.currentTarget) {
        e.preventDefault();
        //其他代码
    }
    e.stopPropagation();
}, false);
```

第 4 步，在 if 语句中，获取被选中图片的 data-name 属性值，然后将'.php'添加到后面拼成一个要访问的页面地址，并将其作为第三个参数传递给 pushState()方法。当然，此处也可以直接使用<a>的 href 属性值。

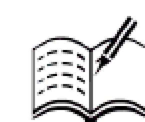
```
var data = e.target.getAttribute('data-name'),
url = data + ".php";
history.pushState(null, null, url);
//此处更改当前的 classes 样式
//然后使用 data 变量的值更新
//并通过 Ajax 请求.content 元素的内容
//最后再更新当前文档的 title
```

> 注意：在真实的示例应用中可能会在 Ajax 请求成功之后才会去修改 URL。

第 5 步，上面代码将真实代码中的内容都替换成注释了，以便读者可以只关注 pushState()方法的



使用。现在单击图片，URL 和 Ajax 请求的内容会被自动更新，但是当单击浏览器工具栏中的“后退”按钮时，并不会回退到之前选中的图片。这里还需要在用户单击“后退”和“前进”按钮时，使用另外一个 Ajax 请求来更新内容，并再一次使用 pushState()方法来更新页面的 URL。这里使用 pushState()方法中的第一个参数（状态对象）来保存状态信息。



Note

```
history.pushState(data, null, url);
```

第 6 步，把上面代码中的 data 参数传递给 popstate 事件处理程序。当浏览器的“后退”和“前进”按钮被单击时，会触发 popstate 事件。

```
window.addEventListener('popstate', function(e) {
    // e.state 表示上一个被单击的图片的 data-attribute
});
```

第 7 步，通过 data 参数可以传递一些有价值的信息，在本示例中将之前选中的图片作为参数传递给 requestContent()方法，在该方法中使用 jQuery 的 load()方法进行一次 Ajax 请求。

```
function requestContent(file){
    $('.content').load(file + ' .content');
}
```

第 8 步，解决了核心技术问题，下面完善 popstate 事件处理程序。

```
window.addEventListener('popstate', function(e){
    var character = e.state;
    if (character == null) {
        removeCurrentClass();
        textWrapper.innerHTML = " ";
        content.innerHTML = " ";
        document.title = defaultTitle;
    } else {
        updateText(character);
        requestContent(character + ".php");
        addCurrentClass(character);
        document.title = "Ghostbuster | " + character;
    }
})
```

第 9 步，完善 index.html 首页内容，该页面除了 HTML 结构，还包含样式表文件：images/style.css、images/style1.css，其中 images/style1.css 导入之后，先隐藏显示，这样能够实现动态显示效果。

```
<link rel="stylesheet" href="images/style1.css" style="display:none !important;">
```

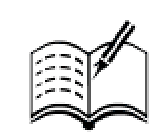
脚本文件 images/app.js，完整代码请参考资源包示例。

第 10 步，设计请求页面，本网站包含 4 个请求页面：zhaozhishu.php、huoyao.php、yinshuashu.php、zhinanzhen.php，虽然都是 php 页面，但是都以静态 HTML 代码设计，如果读者没有 PHP 服务器，可以把它们全部改为.html 静态页面，同时需要在 index.html 页面中修改<a>中的 href 属性值，另外还需要修改 JavaScript 脚本中下面代码句中的".php"：

```
var data = e.target.getAttribute('data-name'),
    url = data + ".php";
```



第 11 步，4 个请求页面的结构相同，内容略有变化，以 zhaozhishu.php 文档为例，其 HTML 结构如下所示，其他页面结构就不再展开，请参考光盘示例。



Note

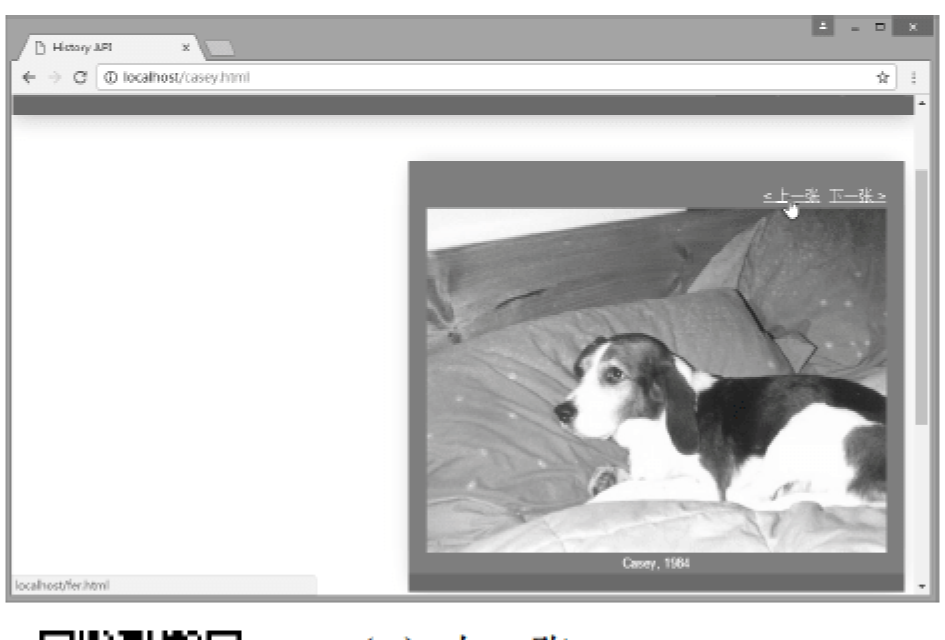
```
<div id="demo-top-bar">
    <div id="demo-bar-inside">
        <h2 id="demo-bar-badge"> <a href="/">中国四大发明</a> </h2>
        <div id="demo-bar-buttons"> </div>
    </div>
</div>
<div class="page-wrap">
    <div class="gallery"> <img src="images/zhaozhishu.png" alt="造纸术" class="zhaozhishu"/> </div>
    <h1>造纸术</h1>
    <div class="content">
        <p>造纸术是中国四大发明之一，纸是中国古代劳动人民长期经验的积累和智慧的结晶，人类文明史上的一项杰出的发明创造。中国是世界上最早养蚕织丝的国家。中国古代劳动人民以上等蚕茧抽丝织绸，剩下的恶茧、病茧等则用漂絮法制取丝绵。漂絮完毕，篾席上会遗留一些残絮。当漂絮的次数多了，篾席上的残絮便积成一层纤维薄片，经晾干之后剥离下来，可用于书写。这种漂絮的副产物数量不多，在古书上称它为赫蹏或方絮。这表明了中国古代造纸术的起源同丝絮有着渊源关系。</p>
        <small><a href="http://baike.baidu.com/">来源：百度百科</a></small> </div>
</div>
```

上面示例简单通过 jQuery 来动态加载内容，用户可以在 pushState()方法中通过状态对象参数传递一些更复杂的信息。

视频讲解

10.2.3 设计无刷新灯箱广告

本例设计一个简单的灯箱广告，它使用 History API 展示了一个图片预览模式：一个具有相关性的图片无刷新访问。在支持的浏览器中浏览，单击下一张图片画廊的链接将更新照片和更新 URL 地址，没有引发全页面刷新。在不支持的浏览器，或者当用户禁用了脚本时，导航链接只是作为普通链接，会打开一个新的页面，整页刷新。整个示例演示效果如图 10.7 所示。



（a）上一张



（b）下一张

图 10.7 无刷新图片画廊演示效果





线上阅读

具体操作步骤请扫码学习。



视频讲解

10.2.4 设计可后退画板

本例利用 History API 的状态对象，实时记录用户的每一次操作，把每一次操作信息传递给浏览



器的历史记录保存起来，这样当用户单击浏览器的“后退”按钮时，会逐步恢复前面的操作状态，从而实现历史恢复功能，演示效果如图 10.8 所示。





（a）绘制文字



（b）恢复前面的绘制

图 10.8　设计历史恢复效果



线 上 阅 读

在示例页面中显示一个 canvas 元素，用户可以在该 canvas 元素中随意使用鼠标绘画，当用户单击一次或连续单击浏览器的“后退”按钮时，可以撤销当前绘制的最后一笔或多笔，当用户单击一次或连续单击浏览器的“前进”按钮时，可以重绘当前书写或绘制的最后一笔或多笔。

具体操作步骤请扫码学习。

10.3　在 线 练 习



在 线 练 习

使用 HTML5 History API 修改站点的 URL。



第11章 文件操作

HTML5 新增 FileReader API 和 FileSystem API。其中 FileReader API 负责读取文件内容，FileSystem API 负责本地文件系统的有限操作。另外，HTML5 增强了 HTML4 的文件域功能，允许提交多个文件，本章将围绕这些 API，详细介绍 HTML5 的本地文件操作功能。

【学习重点】

- 使用 FileList 对象。
- 使用 Blob 对象。
- 使用 FileReader 对象。
- 使用 ArrayBuffer 对象和 ArrayBufferView 对象。
- 使用 FileSystem API。





视频讲解



Note

11.1　FileList

HTML5 在 HTML4 文件域基础上为 File 控件新添 multiple 属性，允许用户在一个 File 控件内选择和提交多个文件。

【示例 1】下面示例设计在文档中插入一个文件域，允许用户同时提交多个文件。

```
<input type="file" multiple>
```

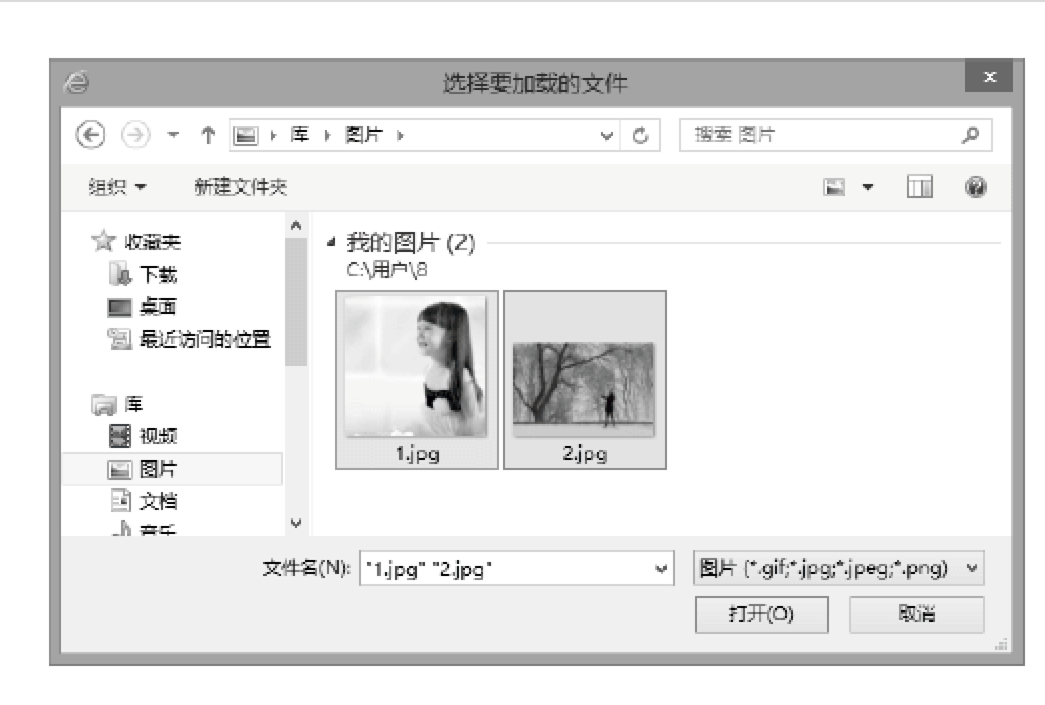
为了方便用户在脚本中访问这些将要提交的文件，HTML5 新增了 FileList 和 File 对象。

☑　FileList：表示用户选择的文件列表。

☑　File：表示 File 控件内的每一个被选择的文件对象。FileList 对象为这些 File 对象的列表，代表用户选择的所有文件。

【示例 2】下面示例演示了如何使用 FileList 和 File 对象访问用户提交的文件名称列表，演示效果如图 11.1 所示。

```
<script>
function ShowFileName(){
    //document.getElementById("file").files 返回 FileList 对象
    for(var i=0;i<document.getElementById("file").files.length;i++) {
        var file = document.getElementById("file").files[i];    //获取每个选择的 File 对象
        console.log(file.name);                                  //在控制台显示每个文件的名称
    }
}
</script>
<input type="file" id="file" multiple>
<input type="button" onclick="ShowFileName();" value="文件上传"/>
```



（a）选择多个文件



（b）在控制台显示提示信息



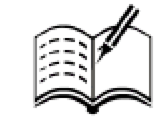
示例效果

图 11.1　使用 FileList 和 File 对象获取提交文件信息

提示：File 对象包含两个属性：name 属性表示文件名，但不包括路径；lastModifiedDate 属性表示文件的最后修改日期。




11.2　Blob



Note



视频讲解

HTML5 的 Blob 对象用于存储二进制数据，还可以设置存储数据的 MINE 类型，其他 HTML5 二进制对象继承 Blob 对象。

11.2.1　访问 Blob

Blob 对象包含两个属性：

☑　size：表示一个 Blob 对象的字节长度。

☑　type：表示 Blob 的 MIME 类型，如果为未知类型，则返回一个空字符串。

【示例 1】下面示例演示了如何获取文件域中第一个文件的 Blob 对象，并访问该文件的长度和文件类型，演示效果如图 11.2 所示。

```
<script>
function ShowFileType(){
    var file = document.getElementById("file").files[0];  //获取用户选择的第一个文件
    console.log( file.size );                             //显示文件字节长度
    console.log( file.type);                              //显示文件类型
}
</script>
<input type="file" id="file" multiple>
<input type="button" onclick="ShowFileType();" value="文件上传"/>
```

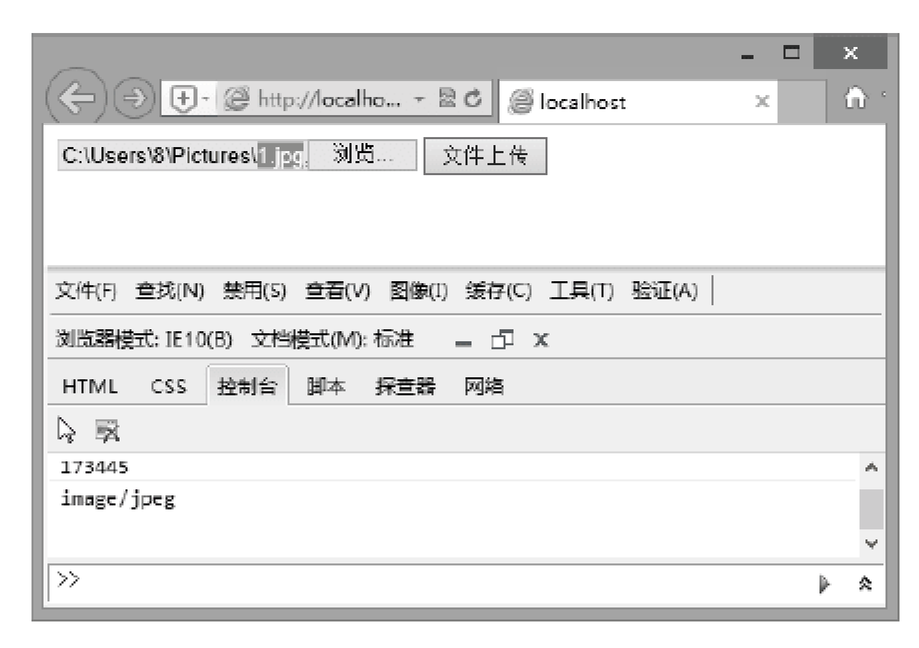


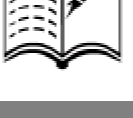
图 11.2　在控制台显示第一个选取文件的大小和类型

注意：对于图像类型的文件，Blob 对象的 type 属性都是以“image/”开头的，后面是图像类型。

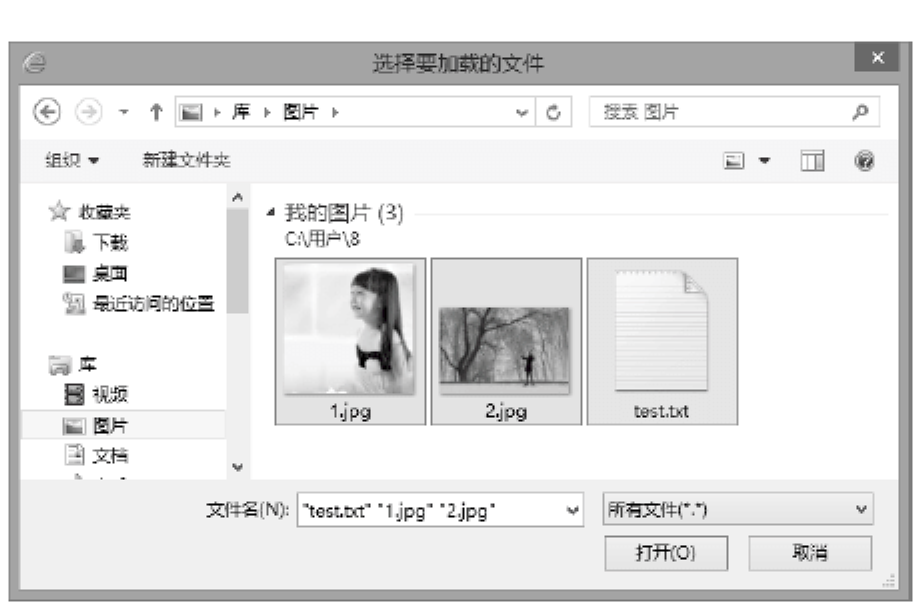
【示例 2】下面示例利用 Blob 的 type 属性，判断用户选择的文件是否为图像文件。如果在批量上传时只允许上传图像文件，可以检测每个文件的 type 属性值，当提交非图像文件时，弹出错误提示信息，并停止后面的文件上传，或者跳过不上传该文件，演示效果如图 11.3 所示。

```
<script>
function fileUpload(){
    var file;
    for(var i=0;i<document.getElementById("file").files.length;i++){
        file = document.getElementById("file").files[i];
        if(!/image\/\w+/.test(file.type)){
```

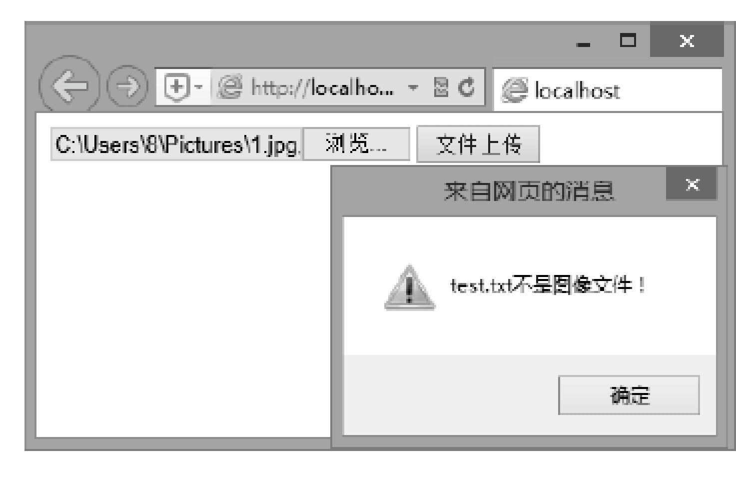





```
                alert(file.name+"不是图像文件！");
                continue;
            } else{
                //此处加入文件上传的代码
                alert(file.name+"文件已上传");
            }
        }
    }
</script>
<input type="file" id="file" multiple>
<input type="button" onclick="fileUpload();" value="文件上传"/>
```



（a）提交多个文件



（b）错误提示信息



示例效果

图 11.3　对用户提交文件进行过滤

【拓展】

HTML5 为 file 控件新添加 accept 属性，设置 file 控件只能接受某种类型的文件。目前主流浏览器对其支持还不统一、不规范，部分浏览器仅限于打开文件选择窗口时，默认选择文件类型。

```
<input type="file" id="file" accept="image/*" />
```

11.2.2　创建 Blob



视频讲解

创建 Blob 对象的基本方法如下：

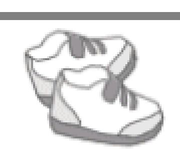
```
var blob = new Blob(blobParts, type);
```

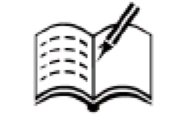
参数说明如下：

☑　blobParts：可选参数，数组类型，其中可以存放任意个以下类型的对象，这些对象中所携带的数据将被依序追加到 Blob 对象中。

- ➢　ArrayBuffer 对象。
- ➢　ArrayBufferView 对象。
- ➢　Blob 对象。
- ➢　String 对象。

☑　type：可选参数，字符串型，设置被创建的 Blob 对象的 type 属性值，即定义 Blob 对象的 MIME 类型。默认参数值为空字符串，表示未知类型。





Note

提示：当创建 Blob 对象时，可以使用两个可选参数。如果不使用任何参数，创建的 Blob 对象的 size 属性值为 0，即 Blob 对象的字节长度为 0，代码如下所示。

```
var blob = new Blob();
```

【示例 1】下面代码演示了如何设置第一个参数。

```
var blob = new Blob(["4234" + "5678"]);
var shorts = new Uint16Array(buffer, 622, 128);
var blobA = new Blob([blob, shorts]);
var bytes = new Uint8Array(buffer, shorts.byteOffset + shorts.byteLength);
var blobB = new Blob([blob, blobA, bytes])
var blobC = new Blob([buffer, blob, blobA, bytes]);
```

注意：上面代码用到了 ArrayBuffer 对象和 ArrayBufferView 对象，后面将详细介绍这两个对象。

【示例 2】下面代码演示了如何设置第二个参数。

```
var blob = new Blob(["4234" + "5678"], {type: "text/plain"});
var blob = new Blob(["4234" + "5678"], {type: "text/plain; charset=UTF-8"});
```

提示：为了安全起见，在创建 Blob 对象之前，可以先检测一下浏览器是否支持 Blob 对象。

```
if(!window.Blob)
    alert ("您的浏览器不支持 Blbo 对象。");
else
    var blob = new Blob(["4234" + "5678"], {type: "text/plain"});
```

目前，各主流浏览器的最新版本都支持 Blob 对象。

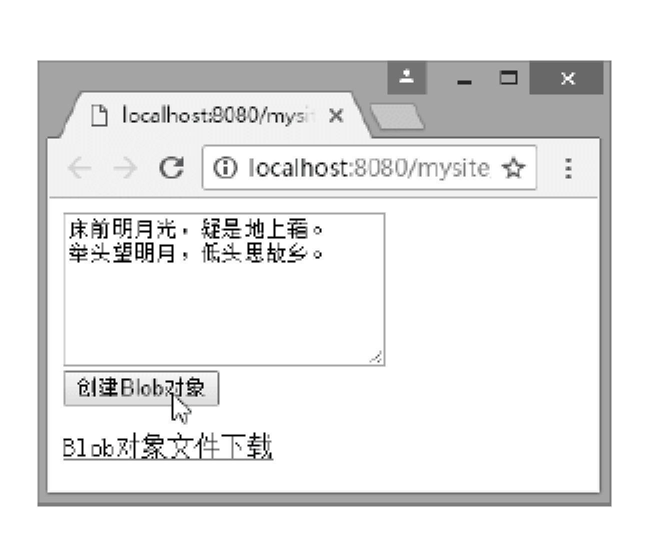
【示例 3】下面示例完整地演示了如何创建一个 Blob 对象。

在页面中设计一个文本区域和一个按钮，当在文本框中输入文字，然后单击“创建 Blob 对象”按钮后，JavaScript 脚本根据用户输入文字创建二进制对象，再根据该二进制对象中的内容创建 URL 地址，最后在页面底部动态添加一个“Blob 对象文件下载”链接，单击该链接可以下载新创建的文件，使用文本文件打开，其内容为用户在文本框中输入的文字，如图 11.4 所示。

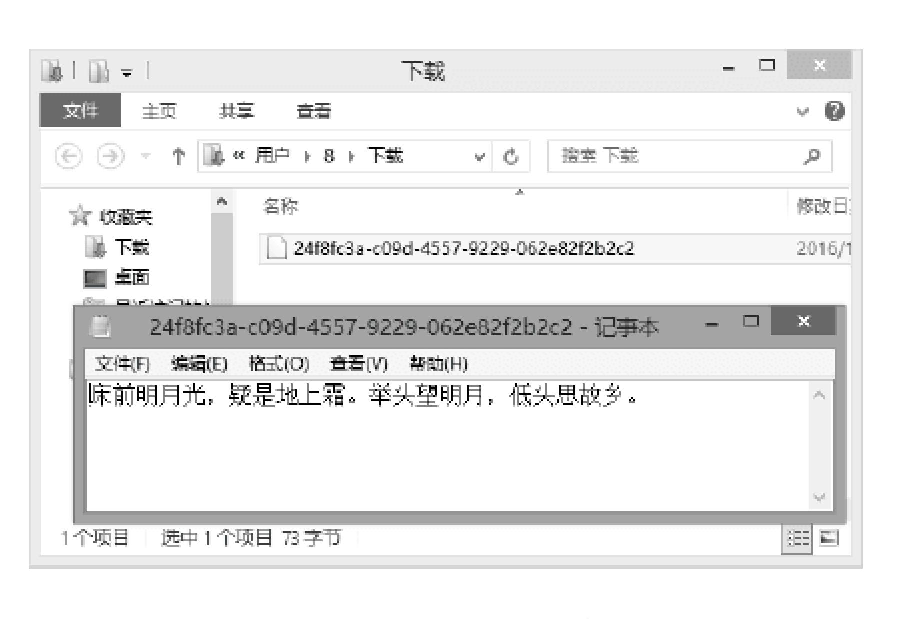
```
<script>
function test(){
    var text = document.getElementById("textarea").value;
    var result = document.getElementById("result");
    //创建 Blob 对象
    if(!window.Blob)
        result.innerHTML="浏览器不支持 Blob 对象。";
    else
        var blob =new Blob([text]);    //Blob 中数据为文字时默认使用 utf8 格式
    //通过 createObjectURL 方法创建文字链接
    if (window.URL) {
        result.innerHTML = '<a download href="' +window.URL.createObjectURL(blob) + '" target="_blank">Blob 对象文件下载</a>';
    }
}
</script>
```



```
<textarea id="textarea"></textarea><br />
<button onclick="test()">创建 Blob 对象</button>
<p id="result"></p>
```



（a）创建 Blob 文件



（b）查看文件信息



示 例 效 果

图 11.4　创建和查看 Blob 文件信息

在动态生成的<a>标签中包含 download 属性，它设置超链接为文件下载类型。

【拓展】

HTML5 支持 URL 对象，通过该对象的 createObjectURL 方法可以根据一个 Blob 对象的二进制数据创建一个 URL 地址，并返回该地址，当用户访问该 URL 地址时，可以直接下载原始二进制数据。



视 频 讲 解

11.2.3　截取 Blob

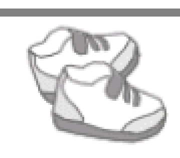
Blob 对象包含 slice()方法，它可以从 Blob 对象中截取一部分数据，然后将这些数据创建为一个新的 Blob 对象并返回，用法如下。

```
var newBlob = blob.slice(start, end, contentType);
```

参数说明如下：

☑ start：可选参数，整数值，设置起始位置。
  - 如果值为 0，表示从第一个字节开始复制数据。
  - 如果值为负数，且 Blob 对象的 size 属性值+start 参数值大于等于 0，则起始位置为 Blob 对象的 size 属性值+start 参数值。
  - 如果值为负数，且 Blob 对象的 size 属性值+start 参数值小于 0，则起始位置为 Blob 对象的起点位置。
  - 如果值为正数，且大于等于 Blob 对象的 size 属性值，则起始位置为 Blob 对象的 size 属性值。
  - 如果值为正数，且小于 Blob 对象的 size 属性值，则起始位置为 start 参数值。

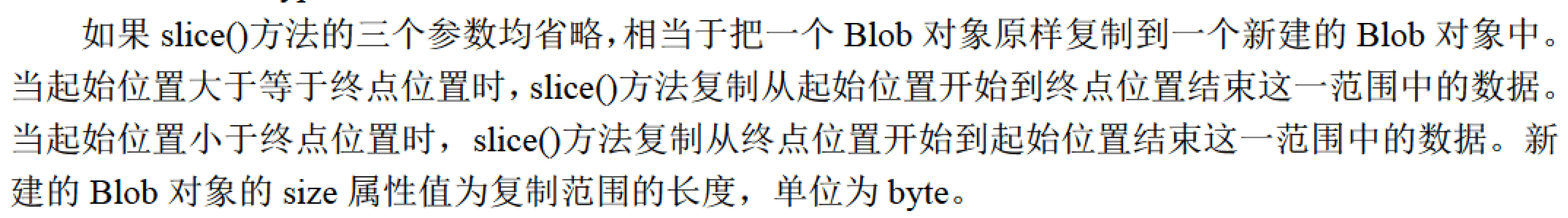
☑ end：可选参数，整数值，设置终点位置。
  - 如果忽略该参数，则终点位置为 Blob 对象的结束位置。
  - 如果值为负数，且 Blob 对象的 size 属性值+end 参数值大于等于 0，则终点位置为 Blob 对象的 size 属性值+end 参数值。
  - 如果值为负数，且 Blob 对象的 size 属性值+end 参数值小于 0，则终点位置为 Blob 对象的起始位置。
  - 如果值为正数，且大于等于 Blob 对象的 size 属性值，则终点位置为 Blob 对象的 size



属性值。

➢ 如果值为正数，且小于 Blob 对象的 size 属性值，则终点位置为 end 参数值。

☑ contentType：可选参数，字符串值，指定新建 Blob 对象的 MIME 类型。



如果 slice()方法的三个参数均省略，相当于把一个 Blob 对象原样复制到一个新建的 Blob 对象中。当起始位置大于等于终点位置时，slice()方法复制从起始位置开始到终点位置结束这一范围中的数据。当起始位置小于终点位置时，slice()方法复制从终点位置开始到起始位置结束这一范围中的数据。新建的 Blob 对象的 size 属性值为复制范围的长度，单位为 byte。

【示例】下面示例演示了 Blob 对象的 slice()方法应用。

```
<input type="file" id="file" multiple>
<input type="button" onclick="ShowFileType();" value="文件上传"/>
<script>
var file = document.getElementById("file").files[0];
if(file){
    var file1 = file.slice();                          //复制 File 对象
    var file2 = file.slice(0,file.size);               //复制 File 对象
    var file3 = file.slice(-(Math.round(file.size/2))); //复制 File 对象的后半部分
    var file4 = file.slice(0, Math.round(file.size/2)); //复制 File 对象的前半部分
    //复制 File 对象，从开始处复制到结束处之前的 150 个字节处，并设置 MIME 类型
    var file5 = file.slice(0,-150, "application/plain");
}
</script>
```



视频讲解

11.2.4 保存 Blob

HTML5 支持在 indexedDB 数据库中保存 Blob 对象。

提示：目前 Chrome 37+、Firefox 17+、IE 10+和 Opera 24+支持该功能。

【示例】下面示例设计在页面中显示一个文件控件和一个按钮，通过文件控件选取文件后，单击按钮 JavaScript 脚本将把用户选取的文件保存到 indexedDB 数据库中。

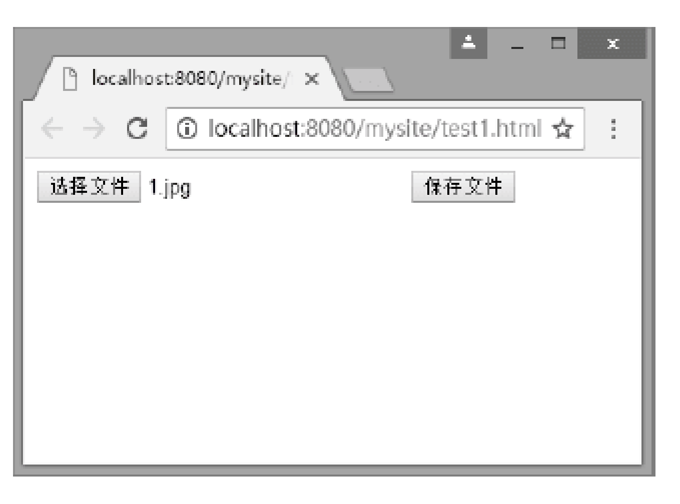
```
<input type="file" id="file" multiple>
<input type="button" onclick="saveFile();" value="保存文件"/>
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange || window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor || window.msIDBCursor;
var dbName = 'test';           //数据库名
var dbVersion = 20170202;      //版本号
var idb;
var dbConnect = indexedDB.open(dbName, dbVersion);
dbConnect.onsuccess = function(e){ idb = e.target.result; }
dbConnect.onerror = function(){alert('数据库连接失败'); };
dbConnect.onupgradeneeded = function(e){
    idb = e.target.result;
    idb.createObjectStore('files');
};
```



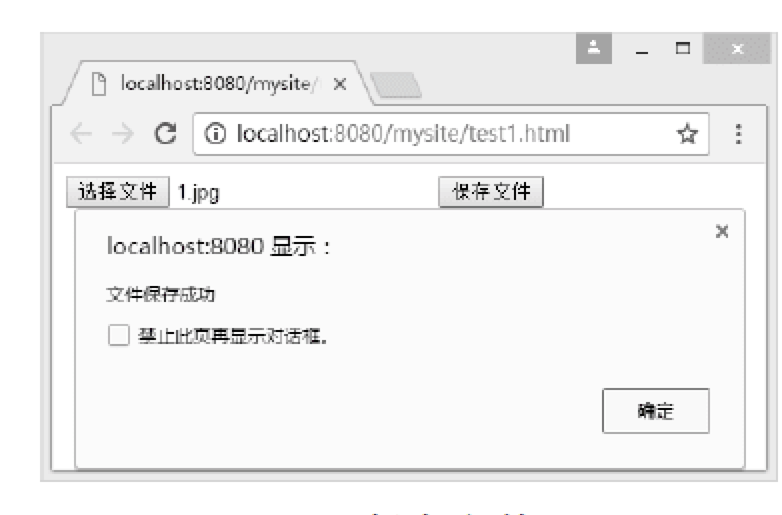



```
function saveFile(){
    var file = document.getElementById("file").files[0];       //得到用户选择的第一个文件
    var tx = idb.transaction(['files'],"readwrite");            //开启事务
    var store = tx.objectStore('files');
    var req = store.put(file,'blob');
    req.onsuccess = function(e){ alert("文件保存成功"); };
    req.onerror = function(e){ alert("文件保存失败");};
}
</script>
```

在浏览器中预览，页面中显示一个文件控件和一个按钮，通过文件控件选取文件，然后单击“保存文件”按钮，JavaScript 将把用户选取文件保存到 indexedDB 数据库中，保存成功后弹出提示对话框，如图 11.5 所示。



（a）选择文件



（b）保存文件



示 例 效 果

图 11.5　保存 Blob 对象应用

11.3　FileReader

FileReader 能够把文件读入内存，并且读取文件中的数据。目前，Firefox 3.6+、Chrome 6+、Safari 5.2+、Opera 11+和 IE 10+版本浏览器都支持 FileReader 对象。

11.3.1　读取文件



视 频 讲 解

使用 FileReader 对象之前，需要实例化 FileReader 类型，代码如下所示：

```
if(typeof FileReader == "undefined"){alert("当前浏览器不支持 FileReader 对象");}
else{ var reader = new FileReader();}
```

FileReader 对象包含 5 个方法，其中前 4 个用以读取文件，另一个用来中断读取操作。

☑ readAsText(Blob, type)：将 Blob 对象或文件中的数据读取为文本数据。该方法包含两个参数，其中第二个参数是文本的编码方式，默认值为 UTF-8。

☑ readAsBinaryString(Blob)：将 Blob 对象或文件中的数据读取为二进制字符串。通常调用该方法将文件提交到服务器端，服务器端可以通过这段字符串存储文件。

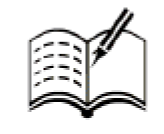
☑ readAsDataURL(Blob)：将 Blob 对象或文件中的数据读取为 DataURL 字符串。该方法就是将数据以一种特殊格式的 URL 地址形式直接读入页面。

☑ readAsArrayBuffer(Blob)：将 Blob 对象或文件中的数据读取为一个 ArrayBuffer 对象。

☑ abort()：不包含参数，中断读取操作。




注意：上述前 4 个方法都包含一个 Blob 对象或 File 对象参数，无论读取成功或失败，都不会返回读取结果，读取结果存储在 result 属性中。



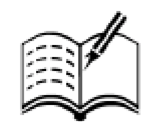
Note

【示例】下面示例演示如何在网页中读取并显示图像文件、文本文件和二进制代码文件。

```
<script>
window.onload = function(){
    var result=document.getElementById("result");
    var file=document.getElementById("file");
    if (typeof FileReader == 'undefined' ){
        result.innerHTML = "<h1>当前浏览器不支持 FileReader 对象</h1>";
        file.setAttribute('disabled', 'disabled' );
    }
}
function readAsDataURL(){ //将文件以 Data URL 形式进行读入页面
    var file = document.getElementById("file").files[0];  //检查是否为图像文件
    if(!/image\/\w+/.test(file.type)){
        alert("提交文件不是图像类型");
        return false;
    }
    var reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = function(e){
        result.innerHTML = '<img src="'+this.result+'" alt=""/>'
    }
}
function readAsBinaryString(){     //将文件以二进制形式进行读入页面
    var file = document.getElementById("file").files[0];
    var reader = new FileReader();
    reader.readAsBinaryString(file);
    reader.onload = function(f){
        result.innerHTML=this.result;
    }
}
function readAsText(){ //将文件以文本形式进行读入页面
    var file = document.getElementById("file").files[0];
    var reader = new FileReader();
    reader.readAsText(file);
    reader.onload = function(f) {
        result.innerHTML=this.result;
    }
}
</script>
<input type="file" id="file" />
<input type="button" value="读取图像" onclick="readAsDataURL()"/>
<input type="button" value="读取二进制数据" onclick="readAsBinaryString()"/>
<input type="button" value="读取文本文件" onclick="readAsText()"/>
<div name="result" id="result"></div>
```

在 Firefox 浏览器中预览，使用 file 控件选择一个图像文件，然后单击“读取图像”按钮，显示





效果如图 11.6 所示；重新使用 file 控件选择一个二进制文件，然后单击“读取二进制数据”按钮，显示效果如图 11.7 所示；最后选择文本文件，单击“读取文本文件”按钮，显示效果如图 11.8 所示。



图 11.6　读取图像文件

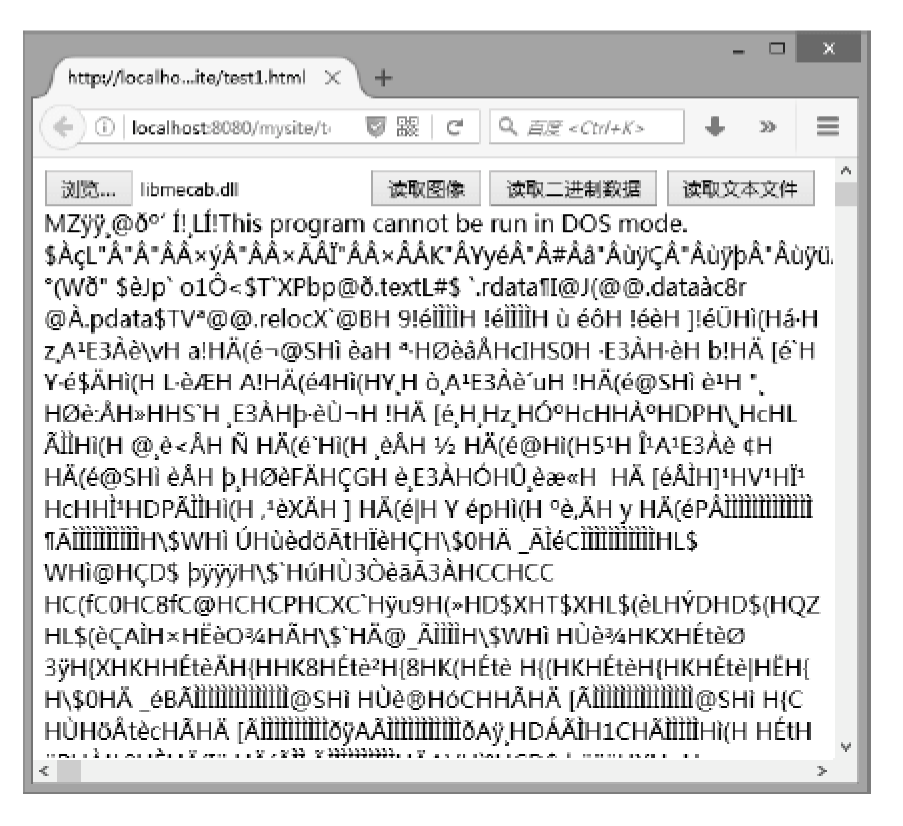


图 11.7　读取二进制文件

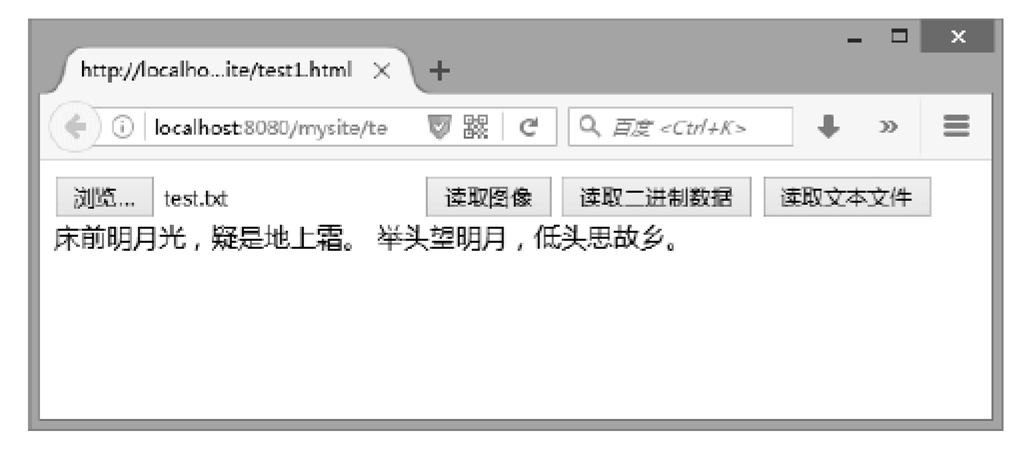


图 11.8　读取文本文件



示 例 效 果

上面示例演示如何读显文件，用户也可以选择不显示，直接提交给服务器，然后保存到文件或数据库中。注意，FileReader 对象读取的数据都保存在 result 属性中。

11.3.2　事件监测



视 频 讲 解

FileReader 对象提供 6 个事件，用于监测文件读取状态，简单说明如下。

☑ onabort：数据读取中断时触发。

☑ onprogress：数据读取中触发。

☑ onerror：数据读取出错时触发。

☑ onload：数据读取成功完成时触发。

☑ onloadstart：数据开始读取时触发。

☑ onloadend：数据读取完成时触发，无论成功或失败。

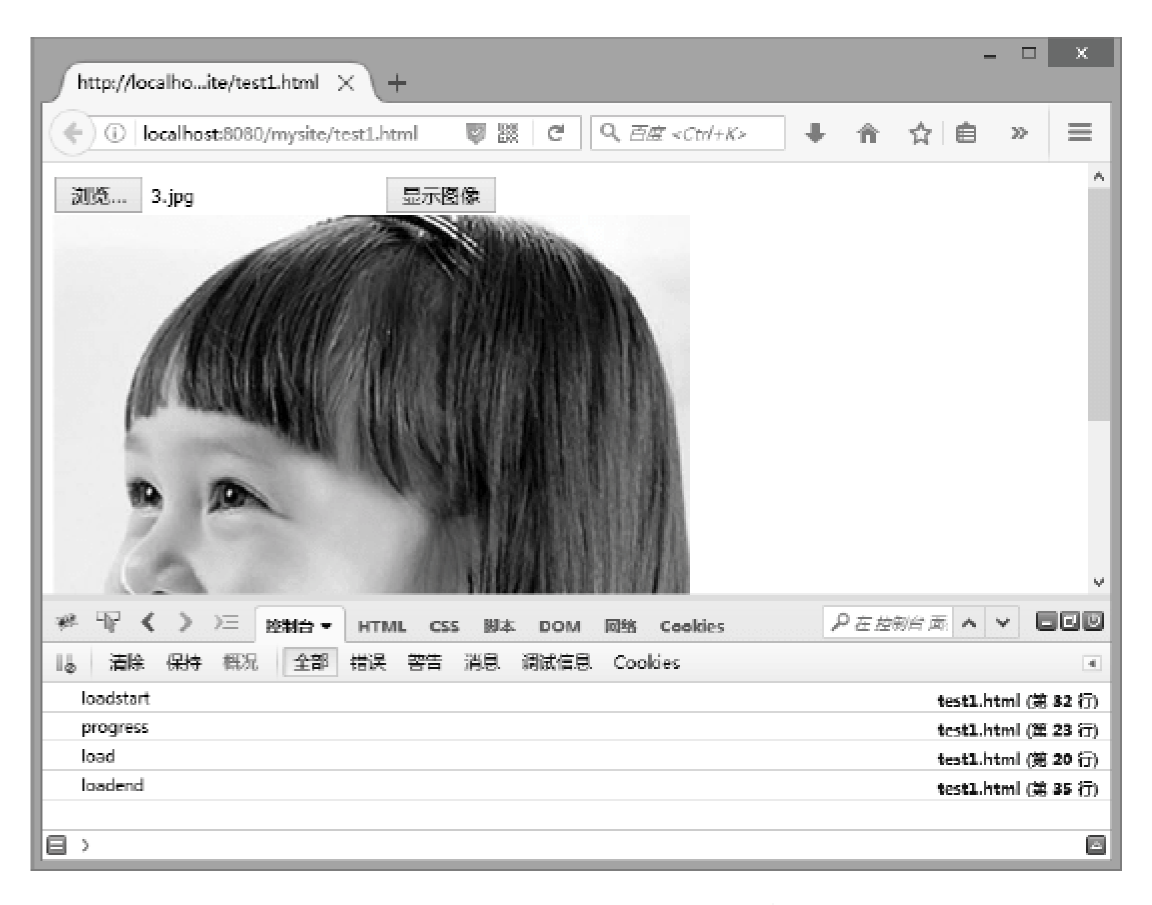
【示例】下面示例设计当使用 FileReader 对象读取文件时会发生一系列事件，在控制台跟踪了读取状态的先后顺序，演示效果如图 11.9 所示。

```
<script>
window.onload = function(){
    var result=document.getElementById("result");
    var file=document.getElementById("file");
    if (typeof FileReader == 'undefined' ){
        result.innerHTML = "<h1>当前浏览器不支持 FileReader 对象</h1>";
```



Note

```
            file.setAttribute('disabled', 'disabled' );
        }
    }
    function readFile(){
        var file = document.getElementById("file").files[0];
        var reader = new FileReader();
        reader.onload = function(e){
            result.innerHTML = '<img src="'+this.result+'" alt=""/>'
            console.log("load");
        }
        reader.onprogress = function(e){ console.log("progress"); }
        reader.onabort = function(e){ console.log("abort"); }
        reader.onerror = function(e){ console.log("error");}
        reader.onloadstart = function(e){ console.log("loadstart");}
        reader.onloadend = function(e){ console.log("loadend"); }
        reader.readAsDataURL(file);
    }
    </script>
    <input type="file" id="file" />
    <input type="button" value="显示图像" onclick="readFile()" />
    <div name="result" id="result"></div>
```





示 例 效 果

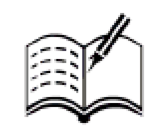
图 11.9　跟踪读取操作

在上面示例中，当单击“显示图像”按钮后，将在页面中读入一个图像文件，同时在控制台可以看到按顺序触发的事件。用户还可以在 onprogress 事件中使用 HTML5 新增元素 progress 显示文件的读取进度。

11.4　ArrayBuffer 和 ArrayBufferView

HTML5 新增 ArrayBuffer 对象和 ArrayBufferView 对象。ArrayBuffer 对象表示一个固定长度的缓存区，用来存储文件或网络大数据；ArrayBufferView 对象表示将缓存区中的数据转换为各种类型的数值数组。





注意：HTML5 不允许直接对 ArrayBuffer 对象内的数据进行操作，需要使用 ArrayBufferView 对象来读写 ArrayBuffer 对象中的内容。

11.4.1　使用 ArrayBuffer

ArrayBuffer 对象表示一个固定长度的存储二进制数据的缓存区。用户不能直接存取 ArrayBuffer 缓存区中的内容，必须通过 ArrayBufferView 对象来读写 ArrayBuffer 缓存区中的内容。ArrayBuffer 对象包含 length 属性，该属性值表示缓存区的长度。

创建 ArrayBuffer 对象的方法如下：

```
var buffer = new ArrayBuffer(32);
```

参数为一个无符号长整型的整数，用于设置缓存区的长度，单位为 byte。ArrayBuffer 缓存区创建成功之后，该缓存区内存储数据初始化为 0。

提示：目前，Firefox 4+、Opera 11.6+、Chrome 7+、Safari 5.1+、IE 10+等版本浏览器支持 ArrayBuffer 对象。

11.4.2　使用 ArrayBufferView

HTML5 使用 ArrayBufferView 对象以一种标准格式来表示 ArrayBuffer 缓存区中的数据。HTML5 不允许直接使用 ArrayBufferView 对象，而是使用 ArrayBufferView 的子类实例来存取 ArrayBuffer 缓存区中的数据，各种子类说明如表 11.1 所示。

表 11.1　ArrayBufferView 的子类

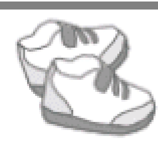
类　型	字 节 长 度	说　明
Int8Array	1	8 位整数数组
Uint8Array	1	8 位无符号整数数组
Uint8ClampedArray	1	8 位无符号整数数组
Int16Array	2	16 位整数数组
Uint16Array	2	16 位无符号整数数组
Int32Array	4	32 位整数数组
Uint32Array	4	32 位无符号整数数组
Float32Array	4	32 位 IEEE 浮点数数组
Float64Array	8	64 位 IEEE 浮点数数组

提示：Uint8ClampedArray 子类用于定义一种特殊的 8 位无符号整数数组，该数组的作用：代替 CanvasPixelArray 数组用于 Canvas API 中。

该数组与普通 8 位无符号整数数组的区别：将 ArrayBuffer 缓存区中的数值进行转换时，内部使用箱位（clamping）算法，而不是模数（modulo）算法。

ArrayBufferView 对象的作用：可以根据同一个 ArrayBuffer 对象创建各种数值类型的数组。

【示例 1】在下面示例代码中，根据相同的 ArrayBuffer 对象，可以创建 32 位的整数数组和 8 位的无符号整数数组。



```
//根据 ArrayBuffer 对象创建 32 位整数数组
var array1 = new Int32Array(Arrayeuffer);
//根据同一个 ArrayBuffer 对象创建 8 位无符号整数数组
var array2 = new Uint8Array(ArrayBuffer);
```

Note

在创建 ArrayBufferView 对象时，除了要指定 ArrayBuffer 缓存区外，还可以使用下面两个可选参数：

☑ byteOffset：为无符号长整型数值，设置开始引用位置与 ArrayBuffer 缓存区第一个字节之间的偏离值，单位为字节。提示，属性值必须为数组中单个元素的字节长度的倍数，省略该参数值时，ArrayBufferView 对象将从 ArrayBuffer 缓存区的第一个字节开始引用。

☑ length：为无符号长整型数值，设置数组中元素的个数。如果省略该参数值，将根据缓存区长度、ArrayBufferView 对象开始引用的位置、每个元素的字节长度自动计算出元素个数。

如果设置了 byteOffset 和 length 参数值，数组从 byteOffset 参数值指定的开始位置开始，长度为：length 参数值所指定的元素个数 × 每个元素的字节长度。

如果忽略了 byteOffset 和 length 参数值，数组将跨越整个 ArrayBuffer 缓存区。

如果省略 length 参数值，数组将从 byteOffset 参数值指定的开始位置到 ArrayBuffer 缓存区的结束位置。

ArrayBufferView 对象包含 3 个属性：

☑ buffer：只读属性，表示 ArrayBuffer 对象，返回 ArrayBufferView 对象引用的 ArrayBuffer 缓存区。

☑ byteOffset：只读属性，表示一个无符号长整型数值，返回 ArrayBufferView 对象开始引用的位置与 ArrayBuffer 缓存区的第一个字节之间的偏离值，单位为字节。

☑ length：只读属性，表示一个无符号长整型数值，返回数组中元素的个数。

【示例 2】下面示例代码演示了如何存取 ArrayBuffer 缓存区中的数据。

```
var byte = array2[4];    //读取第 5 个字节的数据
array2[4] = 1;           //设置第 5 个字节的数据
```



视频讲解

11.4.3 使用 DataView

除了使用 ArrayBufferView 子类外，也可以使用 DataView 类存取 ArrayBuffer 缓存区中的数据。DataView 继承于 ArrayBufferView 类，提供了直接存取 ArrayBuffer 缓存区中数据的方法。

创建 DataView 对象的方法如下：

```
var view = new DataView(buffer, byteOffset, byteLength);
```

参数说明如下：

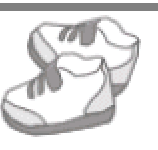
☑ buffer：为 ArrayBuffer 对象，表示一个 ArrayBuffer 缓存区。

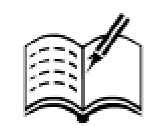
☑ byteOffset：可选参数，为无符号长整型数值，表示 DataView 对象开始引用的位置与 ArrayBuffer 缓存区第一个字节之间的偏离值，单位为字节。如果忽略该参数值，将从 ArrayBuffer 缓存区的第一个字节开始引用。

☑ byteLength：可选参数，为无符号长整型数值，表示 DataView 对象的总字节长度。

如果设置了 byteOffset 和 byteLength 参数值，DataView 对象从 byteOffset 参数值所指定的开始位置开始，长度为 byteLength 参数值所指定的总字节长度。

如果忽略了 byteOffset 和 byteLength 参数值，DataView 对象跨越整个 ArrayBuffer 缓存区。



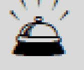


如果省略 byteLength 参数值，DataView 对象将从 byteOffset 参数所指定的开始位置到 ArrayBuffer 缓存区的结束位置。

DataView 对象包含的方法说明如表 11.2 所示。

表 11.2　DataView 对象方法

方　法	说　明
getInt8(byteOffset)	获取指定位置的一个 8 位整数值
getUint8(byteOffset)	获取指定位置的一个 8 位无符号型整数值
getInt16(byteOffset, littleEndian)	获取指定位置的一个 16 位整数值
getUint16(byteOffset, littleEndian)	获取指定位置的一个 16 位无符号型整数值
getUint32(byteOffset, littleEndian)	获取指定位置的一个 32 位无符号型整数值
getFloat32(byteOffset, littleEndian)	获取指定位置的一个 32 位浮点数值
getFloat64(byteOffset, littleEndian)	获取指定位置的一个 64 位浮点数值
setInt8(byteOffset, value)	设置指定位置的一个 8 位整数值
setUint8(byteOffset, value)	设置指定位置的一个 8 位无符号型整数值
setInt16(byteOffset, value, littleEndian)	设置指定位置的一个 16 位整数值
setUint16(byteOffset, value, littleEndian)	设置指定位置的一个 16 位无符号型整数值
setUint32(byteOffset, value, littleEndian)	设置指定位置的一个 32 位无符号型整数值
setFloat32(byteOffset, value, littleEndian)	设置指定位置的一个 32 位浮点数值
setFloat64(byteOffset, value, littleEndian)	设置指定位置的一个 64 位浮点数值



提示：在上述方法中，各个参数说明如下：

☑ byteOffset：为一个无符号长整型数值，表示设置或读取整数所在位置与 DataView 对象对 ArrayBuffer 缓存区的开始引用位置之间相隔多少个字节。

☑ value：为无符号对应类型的数值，表示在指定位置进行设定的整型数值。

☑ littleEndian：可选参数，为布尔类型，判断该整数值的字节序。当值为 true 时，表示以 little-endian 方式设置或读取该整数值(低地址存放最低有效字节)；当参数值为 false 或忽略该参数值时，表示以 big-endian 方式读取该整数值(低地址存放最高有效字节)。

【示例】下面示例演示了如何使用 DataView 对象的相关方法，实现对文件数据进行截取和检测，演示效果如图 11.10 所示。

```
<script>
window.onload = function(){
      var result=document.getElementById("result");
      var file=document.getElementById("file");
      if (typeof FileReader == 'undefined' ){
           result.innerHTML = "<h1>当前浏览器不支持 FileReader 对象</h1>";
           file.setAttribute('disabled', 'disabled' );
      }
}
function file_onchange(){
      var file=document.getElementById("file").files[0];
      if(!/image\/\w+/.test(file.type)){
```



Note

```
            alert("请选择一个图像文件！");
            return;
        }
        var slice=file.slice(0,4);
        var reader = new FileReader();
        reader.readAsArrayBuffer(slice);
        var type;
        reader.onload = function(e){
            var buffer=this.result;
            var view=new DataView(buffer);
            var magic=view.getInt32(0,false);
            if(magic<0)     magic = magic + 0x100000000;
            magic=magic.toString(16).toUpperCase();
            if(magic.indexOf('FFD8FF') >=0)     type="jpg 文件";
            if(magic.indexOf('89504E47') >=0)   type="png 文件";
            if(magic.indexOf('47494638') >=0)     type="gif 文件";
            if(magic.indexOf('49492A00') >=0)     type="tif 文件";
            if(magic.indexOf('424D') >=0)   type="bmp 文件";
            document.getElementById("result").innerHTML ='文件类型为：'+type;
        }
    }
</script>
<input type="file" id="file" onchange="file_onchange()" /><br/>
<output id="result"></output>
```



示例效果

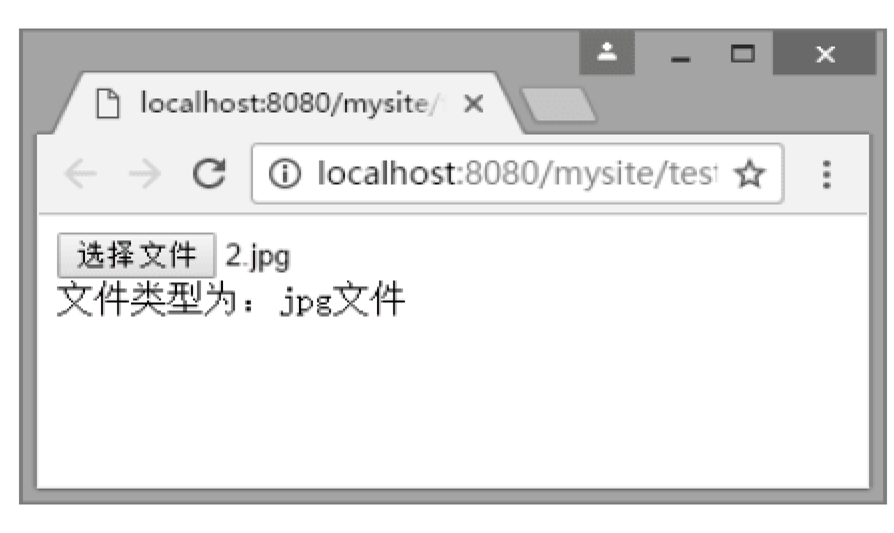


图 11.10　判断选取文件的类型

【操作步骤】

第 1 步，在上面示例中，先在页面中设计一个文件控件。

第 2 步，当用户在浏览器中选取一个图像文件后，JavaScript 先检测文件类型，当为图像文件后，再使用 File 对象的 slice()方法将该文件中前 4 个字节的内容复制到一个 Blob 对象中，代码如下所示。

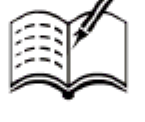
```
var file=document.getElementById("file").files[0];
if(!/image\/\w+/.test(file.type)){
    alert("请选择一个图像文件！");
    return;
}
var slice=file.slice(0,4);
```

第 3 步，新建 FileReader 对象，使用该对象的 readAsArrayBuffer()方法将 Blob 对象中的数据读取为一个 ArrayBuffer 对象，代码如下所示。



```
var reader = new FileReader();
reader.readAsArrayBuffer(slice);
```

第 4 步，读取 ArrayBuffer 对象后，使用 DataView 对象读取该 ArrayBuffer 缓存区中位于开头位置的一个 32 位整数，代码如下所示。



```
reader.onload = function(e){
    var buffer=this.result;
    var view=new DataView(buffer);
    var magic=view.getInt32(0,false);
}
```

第 5 步，最后根据该整数值判断用户选取的文件类型，并将文件类型显示在页面上。

```
if(magic<0)    magic = magic + 0x100000000;
magic=magic.toString(16).toUpperCase();
if(magic.indexOf('FFD8FF') >=0)   type="jpg 文件";
if(magic.indexOf('89504E47') >=0)   type="png 文件";
if(magic.indexOf('47494638') >=0)    type="gif 文件";
if(magic.indexOf('49492A00') >=0)   type="tif 文件";
if(magic.indexOf('424D') >=0)     type="bmp 文件";
document.getElementById("result").innerHTML ='文件类型为：'+type;
```

11.5　FileSystem API

HTML5 的 FileSystem API 可以将数据保存到本地磁盘的文件系统中，实现数据的永久保存。

11.5.1　认识 FileSystem API

FileSystem API 包括两部分内容：一部分内容为除后台线程之外的任何场合使用的异步 API，另一部分内容为后台线程中专用的同步 API。本节仅介绍异步 API 内容。

FileSystem API 具有如下特性：

- ☑ 支持跨域通信，但是每个域的文件系统只能被该域专用，不能被其他域访问。
- ☑ 存储的数据是永久的，不能被浏览器随意删除，但是存储在临时文件系统中的数据可以被浏览器自行删除。
- ☑ 当 Web 应用连续发出多次对文件系统的操作请求时，每一个请求都将得到响应，同时第一个请求中所保存的数据可以被之后的请求立即得到。

目前，只有 Chrome 10+版本浏览器支持 FileSystem API。

11.5.2　访问 FileSystem



使用 window 对象的 requestFileSystem()方法可以请求访问受到浏览器沙箱保护的本地文件系统，用法如下：

```
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
window.requestFileSystem(type, size, successCallback, opt_ errorCallback) ;
```



Note

参数说明如下：

☑ type：设置请求访问的文件系统使用的文件存储空间的类型，取值包括：window.TEMPORARY 和 window.PERSISTENT。当值为 window.TEMPORARY 时，表示请求临时的存储空间，存储在临时存储空间中的数据可以被浏览器自行删除；当值为 window.PERSISTENT 时，表示请求永久存储空间，存储在该空间的数据不能被浏览器在用户不知情的情况下清除，只能通过用户或应用程序来清除，请求永久存储空间需要用户为应用程序指定一定的磁盘配额。

☑ size：设置请求的文件系统使用的文件存储空间的大小，尺寸为 byte。

☑ successCallback：设置请求成功时执行的回调函数，该回调函数的参数为一个 FileSystem 对象，表示请求访问的文件系统对象。

☑ opt_errorCallback：可选参数，设置请求失败时执行的回调函数，该回调函数的参数为一个 FileError 对象，其中存放了请求失败时的各种信息。

FileError 对象包含 code 属性，其值为 FileSystem API 中预定义的常量值，说明如下所示：

☑ FileError.QUOTA_EXCEEDED_ERR：文件系统所使用的存储空间的尺寸超过磁盘配额控制中指定的空间尺寸。

☑ FileError.NOT_FOUND_ERR：未找到文件或目录。

☑ FileError.SECURITY_ERR：操作不当引起安全性错误。

☑ FileError.INVALID_MODIFICATION_ERR：对文件或目录所指定的操作（如文件复制、删除、目录复制、目录删除等处理）不能被执行。

☑ FileError.INVALID_STATE_ERR：指定的状态无效。

☑ FileError. ABORT_ERR：当前操作被终止。

☑ FileError. NOT_READABLE_ERR：指定的目录或文件不可读。

☑ FileError. ENCODING_ERR：文字编码错误。

☑ FileError.TYPE_MISMATCH_ERR：用户企图访问目录或文件，但是用户访问的目录事实上是一个文件或用户访问的文件事实上是一个目录。

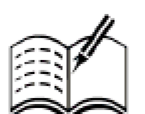
☑ FileError. PATH_EXISTS_ERR：用户指定的路径中不存在需要访问的目录或文件。

【示例】下面示例演示如何在 Web 应用中使用 FileSystem API。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
var fs = null;
if(window.requestFileSystem){
    window.requestFileSystem(window.TEMPORARY, 1024*1024,
    function(filesystem) {
        fs = filesystem;
    }, errorHandler);
}
function errorHandler(FileError) {
    switch (FileError.code) {
        case FileError.QUOTA_EXCEEDED_ERR:
            console.log('文件系统所使用的存储空间的尺寸超过磁盘限额控制中指定的空间尺寸');
            break;
        case FileError.NOT_FOUND_ERR:
            console.log('未找到文件或目录');
            break;
```



```
            case FileError.SECURITY_ERR:
                console.log( '操作不当引起安全性错误');
                break;
            case FileError.INVALID_MODIFICATION_ERR:
                console.log('对文件或目录所指定的操作不能被执行');
                break;
            case FileError.INVALID_STATE_ERR:
                console.log('指定的状态无效');
        };
    }
</script>
```



Note

在上面代码中，先判断浏览器是否支持 FileSystem API，如果支持则调用 window.requestFileSystem() 请求访问本地文件系统，如果请求失败则在控制台显示错误信息。

11.5.3　申请配额



视频讲解

当在磁盘中保存数据时，首先需要申请一定的磁盘配额。在 Chrome 浏览器中，可以通过 window.webkitStorageInfo.requestQuota()方法向用户计算机申请磁盘配额。用法如下：

```
window.webkitStorageInfo.requestQuota(PERSISTENT, 1024*1024,
    //申请磁盘配额成功时执行的回调函数
    function(grantedBytes){
        window.requestFilesystem(PERSISTENT, grantedBytes, onInitFs, errorHandler);
    },
    //申请磁盘配额失败时执行的回调函数
    errorHandler
)
```

该方法包含 4 个参数，说明如下：

第 1 个参数：为 TEMPORARY 或 PERSISTENT。为 TEMPORARY 时，表示为临时数据申请磁盘配额；为 PERSISTENT 时，表示为永久数据申请磁盘配额。

当在用户计算机中保存临时数据，如果其他磁盘空间尺寸不足时，可能会删除应用程序所用磁盘配额中的数据。在磁盘配额中保存数据后，当浏览器被关闭或关闭计算机电源时，这些数据不会丢失。

第 2 个参数：为整数值，表示申请的磁盘空间尺寸，单位为 byte。上面代码将参数值设为 1024×1024，表示向用户计算机申请 1GB 的磁盘空间。

第 3 个参数：为一个函数，表示申请磁盘配额成功时执行的回调函数。在回调函数中可以使用一个参数，参数值为申请成功的磁盘空间尺寸，单位为 byte。

第 4 个参数：为一个函数，表示申请磁盘配额失败时执行的回调函数，该回调函数使用一个参数，参数值为一个 FileError 对象，其中存放申请磁盘配额失败时的各种错误信息。

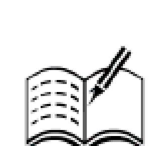


提示： 当 Web 应用首次申请磁盘配额成功后，将立即获得该磁盘配额中指定的磁盘空间，下次使用该磁盘空间时不需要再次申请。

【示例 1】 下面示例演示了如何申请磁盘配额。首先在页面中设计一个文本框，当用户在文本框控件中输入需要申请的磁盘空间尺寸后，JavaScript 向用户申请磁盘配额，申请磁盘配额成功后在页面中显示申请的磁盘空间尺寸。

```
<script>
```





Note

```
function getQuota(){                    //申请磁盘配额
    var size = document.getElementById("capacity").value;
    window.webkitStorageInfo.requestQuota(PERSISTENT,size,
    function(grantedBytes){        //申请磁盘配额成功时执行的回调函数
        var text="申请磁盘配额成功<br>磁盘配额尺寸:"
        var strBytes,intBytes;
        if(grantedBytes>=1024*1024*1024){
            intBytes=Math.floor(grantedBytes/(1024*1024*1024));
            text+=intBytes+"GB ";
            grantedBytes=grantedBytes%(1024*1024*1024);
        }
        if(grantedBytes>=1024*1024){
            intBytes=Math.floor(grantedBytes/(1024*1024));
            text+=intBytes+"MB ";
            grantedBytes=grantedBytes%(1024*1024);
        }
        if(grantedBytes>=1024){
            intBytes=Math.floor(grantedBytes/1024);
            text+=intBytes+"KB ";
            grantedBytes=grantedBytes%1024;
        }
        text+=grantedBytes+"Bytes";
        document.getElementById("result").innerHTML = text;
    },
    errorHandler);              //申请磁盘配额失败时执行的回调函数
}
function errorHandler(FileError) {
    switch (FileError.code) {
        case FileError.QUOTA_EXCEEDED_ERR:
            console.log('文件系统所使用的存储空间的尺寸超过磁盘限额控制中指定的空间尺寸');
            break;
        case FileError.NOT_FOUND_ERR:
            console.log('未找到文件或目录');
            break;
        case FileError.SECURITY_ERR:
            console.log( '操作不当引起安全性错误');
            break;
        case FileError.INVALID_MODIFICATION_ERR:
            console.log('对文件或目录所指定的操作不能被执行');
            break;
        case FileError.INVALID_STATE_ERR:
            console.log('指定的状态无效');
    };
}
</script>
<form>
    <input type="text" id="capacity" value="1024">
    <input type="button" value="申请磁盘配额" onclick="getQuota()">
</form>
<output id="result" ></output>
```



在 Chrome 浏览器中浏览页面，然后在文本框控件中输入 30000，单击“申请磁盘配额”按钮，则 JavaScript 会自动计算出当前磁盘配额空间的大小，如图 11.11 所示。

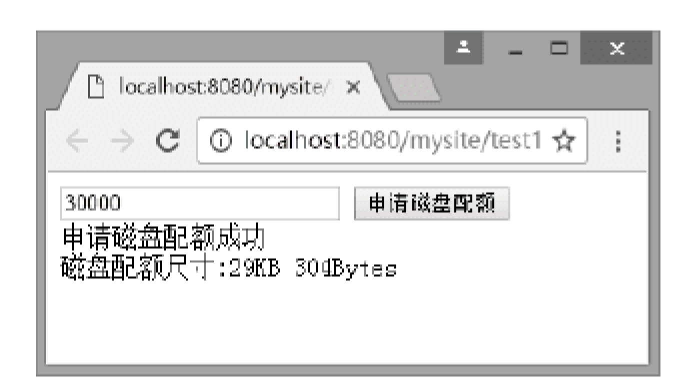


图 11.11　申请磁盘配额

成功申请磁盘配额之后，可以使用 window.webkitStorageInfo.queryUsageAndQuota()方法查询申请的磁盘配额信息，用法如下：

```
window.webkitStorageInfo.queryUsageAndQuota(PERSISTENT,
    //获取磁盘配额信息成功时执行的回调函数
    function(usage,quota)  {
        //代码
    },
    //获取磁盘配额信息失败时执行的回调函数
    errorHandler
);
```

该方法包含三个参数，说明如下：

第 1 个参数：可选 TEMPORARY 或 PERSISTENT 常量值。为 TEMPORARY 时，表示查询保存临时数据用磁盘配额信息；为 PERSISTENT 时，表示查询保存永久数据用磁盘配额信息。

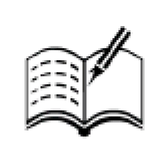
第 2 个参数：函数，表示查询磁盘配额信息成功时执行的回调函数。在回调函数中可以使用两个参数，其中第 1 个参数为磁盘配额中已用磁盘空间尺寸，第 2 个参数表示磁盘配额所指定的全部磁盘空间尺寸，单位为 byte。

第 3 个参数：函数，表示查询磁盘配额信息失败时执行的回调函数。回调函数的参数为一个 FileError 对象，其中存放了查询磁盘配额信息失败时的各种错误信息。

【示例 2】我们看一个查询磁盘配额信息的代码示例。设计在页面中显示一个“查询磁盘配额信息”按钮，当用户单击该按钮时，将查询用户申请的磁盘配额信息。查询成功时将磁盘配额中用户已占用磁盘空间尺寸和磁盘配额的总空间尺寸显示在页面中，演示效果如图 11.12 所示。

```
<script>
function queryQuota(){      //查询磁盘配额信息
    window.webkitStorageInfo.queryUsageAndQuota(PERSISTENT,
    function(usage,quota){  //查询磁盘配额信息成功时执行的回调函数
        var text="查询磁盘配额信息成功<br>已用磁盘空间:"
        var strBytes,intBytes;
        if(usage>=1024*1024*1024){
            intBytes=Math.floor(usage/(1024*1024*1024));
            text+=intBytes+"GB ";
            usage=usage%(1024*1024*1024);
        }
        if(usage>=1024*1024){
            intBytes=Math.floor(usage/1024*1024);
            text+=intBytes+"MB ";
```





Note

```
                usage=usage%1024*1024;
            }
            if(usage>=1024){
                intBytes=Math.floor(usage/1024);
                text+=intBytes+"KB ";
                usage=usage%1024;
            }
            text+=usage+"Bytes";
            text+="<br>磁盘配额的总空间：";
            if(quota>=1024*1024*1024){
                intBytes=Math.floor(quota/(1024*1024*1024));
                text+=intBytes+"GB ";
                quota=quota%(1024*1024*1024);
            }
            if(quota>=1024*1024){
                intBytes=Math.floor(quota/(1024*1024));
                text+=intBytes+"MB ";
                quota=quota%(1024*1024);
            }
            if(quota>=1024){
                intBytes=Math.floor(quota/1024);
                text+=intBytes+"KB ";
                quota=quota%1024;
            }
            text+=quota+"Bytes";
            document.getElementById("result").innerHTML = text;
        },
        errorHandler);      //申请磁盘配额失败时执行的回调函数
    }
    function errorHandler(FileError) {
        //参考上面示例代码中 errorHandler()
    }
    </script>
    <h1>查询磁盘配额信息</h1>
    <input type="button" value="查询磁盘配额信息" onclick="queryQuota()">
    <output id="result" ></output>
```



示例效果



图 11.12　查询磁盘配额信息

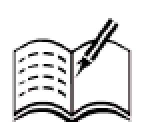


视频讲解

11.5.4　新建文件

创建文件的操作思路：当用户调用 requestFileSystem()方法请求访问本地文件系统时，如果请求





成功，则执行一个回调函数，这个回调函数中包含一个参数，它指向可以获取的文件系统对象，该文件系统对象包含一个 root 属性，属性值为一个 DirectoryEntry 对象，表示文件系统的根目录对象。在请求成功时执行的回调函数中，可以通过文件系统的根目录对象的 getFile()方法在根目录中创建文件。

getFile()方法包含 4 个参数，简单说明如下：

第 1 个参数：为字符串值，表示需要创建或获取的文件名。

第 2 个参数：为一个自定义对象。当创建文件时，必须将该对象的 create 属性值设为 true；当获取文件时，必须将该对象的 create 属性值设为 false；当创建文件时，如果该文件已存在，则覆盖该文件；如果该文件已存在，且被使用排他方式打开，则抛出错误。

第 3 个参数：为一个函数，代表获取文件或创建文件成功时执行的回调函数，在回调函数中可以使用一个参数，参数值为一个 FileEntry 对象，表示成功创建或获取的文件。

第 4 个参数：为一个函数，代表获取文件或创建文件失败时执行的回调函数，参数值为一个 FileError 对象，其中存放了获取文件或创建文件失败时的各种错误信息。

FileEntry 对象表示受到沙箱保护的文件系统中每一个文件。该对象包含如下属性：

- ☑ isFile：区分对象是否为文件。属性值为 true，表示对象为文件；属性值为 false，表示该对象为目录。
- ☑ isDirectory：区分对象是否为目录。属性值为 true，表示对象为目录；属性值为 false，表示该对象为文件。
- ☑ name：表示该文件的文件名，包括文件的扩展名。
- ☑ fullPath：表示该文件的完整路径。
- ☑ filesystem：表示该文件所在的文件系统对象。

另外，FileEntry 对象包括 remove()（删除）、moveTo()（移动）、copyTo()（复制）等方法。

【示例】下面示例演示了创建文件的基本方法。在页面中设计两个文本框和一个“创建文件”按钮，其中一个文本框控件用于输入文件名，另一个文本框控件用于输入文件大小，单位为 byte，用户输入文件名及文件大小后，单击“创建文件”按钮，JavaScript 会在文件系统中的根目录下创建文件，并将创建的文件信息显示在页面中，如图 11.13 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function createFile(){                          //创建文件
    var size = document.getElementById("FileSize").value;
    window.requestFileSystem( PERSISTENT,   size,
        function(fs){                           //请求文件系统成功时所执行的回调函数
            var filename = document.getElementById("FileName").value;
            fs.root.getFile(                    //创建文件
                filename,
                { create: true },
                function(fileEntry){  //创建文件成功时所执行的回调函数
                    var text = "完整路径："+fileEntry.fullPath+"<br>";
                    text += "文  件  名："+fileEntry.name+"<br>";
                    document.getElementById("result").innerHTML = text;
                },
                errorHandler            //创建文件失败时所执行的回调函数
            );
        },
        errorHandler                    //请求文件系统失败时所执行的回调函数
```



```
        );
    }
    function errorHandler(FileError) { //省略代码}
    </script>
    <h1>创建文件</h1>
    文 件 名：<input type="text" id="FileName" value="test.txt"><br/><br/>
    文件大小：<input type="text" id="FileSize" value="1024"/>Bytes<br/><br/>
    <input type="button" value="创建文件" onclick="createFile()"><br/><br/>
    <output id="result" ></output>
```



Note



示例效果

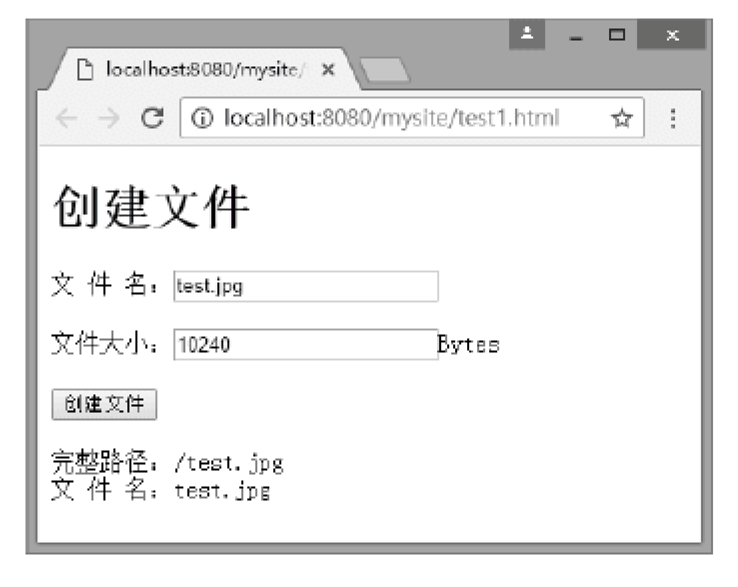


图 11.13　创建文件

注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。



视频讲解

11.5.5　写入数据

HTML5 使用 FileWriter 和 FileWriterSync 对象执行文件写入操作，其中 FileWriterSync 对象用于在后台线程中进行文件的写操作，FileWriter 对象用于除后台线程之外任何场合进行写操作。

在 FileSystem API 中，当使用 DirectoryEntry 对象的 getFile()方法成功获取一个文件对象之后，可以在获取文件对象成功时所执行的回调函数中，利用文件对象的 createWriter()方法创建 FileWriter 对象。

createWriter()方法包含两个参数，分别为创建 FileWriter 对象成功时执行的回调函数和失败时执行的回调函数。在创建 FileWriter 对象成功时执行的回调函数中，包含一个参数，它表示 FileWriter 对象。

使用 FileWriter 对象的 write()方法在获取到的文件中写入二进制数据，用法如下。

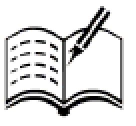
```
fileWriter.write(data);
```

参数 data 为一个 Blob 对象，表示要写入的二进制数据。

使用 FileWriter 对象的 writeend 和 error 事件可以进行监听，在事件回调函数中可以使用一个对象，它表示被触发的事件对象。

【示例】以 11.5.4 节示例为基础，对 createFile()函数进行修改，当用户单击“创建文件”按钮时，首先创建一个文件，在创建文件成功时执行的回调函数中创建一个 Blob 对象，并在其中写入'Hello, World'文字，当写文件操作成功时在页面中显示“写文件操作成功”文字，当写文件操作失败时在页面中显示“写文件操作失败”文字，如图 11.14 所示。





```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function createFile(){          //写入文件操作
    var size = document.getElementById("FileSize").value;
    window.requestFileSystem( PERSISTENT, size,
        function(fs){          //请求文件系统成功时所执行的回调函数
            var filename = document.getElementById("FileName").value;
            fs.root.getFile(filename,          //创建文件
                {create: true},
                function(fileEntry) {
                    fileEntry.createWriter(function(fileWriter) {
                        fileWriter.onwriteend = function(e) {
                            document.getElementById("result").innerHTML ='写文件操作结束';
                        };
                        fileWriter.onerror = function(e) {
                            document.getElementById("result").innerHTML='写文件操作失败: ';
                        };
                        var blob = new Blob(['Hello, World']);
                        fileWriter.write(blob);
                    }, errorHandler);
                }, errorHandler);
        },
        errorHandler          //请求文件系统失败时所执行的回调函数
    );
}
function errorHandler(FileError) { //省略代码}
</script>

<h1>创建文件</h1>
文 件 名：<input type="text" id="FileName" value="test.txt"><br/><br/>
文件大小：<input type="text" id="FileSize" value="1024"/>Bytes<br/><br/>
<input type="button" value="创建文件" onclick="createFile()"><br/>
<output id="result" ></output>
```





示 例 效 果

图 11.14　写入文件

> 注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。





视频讲解



Note

11.5.6 添加数据

向文件添加数据与创建文件并写入数据操作类似，区别在于在获取文件之后，首先需要使用 FileWriter 对象的 seek()方法将文件读写位置设置到文件底部，用法如下。

```
fileWriter.seek(fileWriter.length);
```

参数值为长整型数值。当值为正值时，表示文件读写位置与文件开头处之间的距离，单位为 byte（字节数）；当值为负值时，表示文件读写位置与文件结尾处之间的距离。

【示例】下面示例演示了如何向指定文件添加数据。在页面中设计一个用于输入文件名的文本框和一个“添加数据”按钮，当用户在文件名文本框中输入文件名后，单击“添加数据”按钮，将在该文件中添加“新数据”文字，追加成功后在页面中显示“添加数据成功”提示信息，演示效果如图 11.15 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function addData(){    //向文件中添加数据
    window.requestFileSystem( PERSISTENT, 1024,
        function(fs){ //请求文件系统成功时所执行的回调函数
            var filename = document.getElementById("fileName").value;
            fs.root.getFile(filename,   //创建文件
                {create:false},
                function(fileEntry) {
                    fileEntry.createWriter(function(fileWriter) {
                        fileWriter.onwriteend = function(e) {
                            document.getElementById("result").innerHTML ='添加数据成功';
                        };
                        fileWriter.onerror = function(e) {
                            document.getElementById("result").innerHTML='添加数据失败: ';
                        };
                        fileWriter.seek(fileWriter.length);
                        var blob = new Blob(['新数据']);
                        fileWriter.write(blob);
                    }, errorHandler);
                }, errorHandler);
        },
        errorHandler        //请求文件系统失败时所执行的回调函数
    );
}
function errorHandler(FileError) { //省略代码  }
</script>
<h1>添加数据</h1>
文件名：<input type="text" id="fileName" value="test.txt"><br/><br/>
<input type="button" value="添加数据" onclick="addData()"><br/>
<output id="result" ></output>
```

注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。

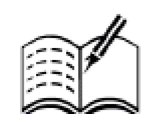




图 11.15 添加数据




示例效果



Note



视频讲解

11.5.7 读取数据

在 FileSystem API 中，使用 FileReader 对象可以读取文件，详细介绍可以参考 11.3 节内容。

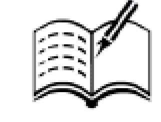
在文件对象（FileEntry）的 file()方法中包含两个参数，分别表示获取文件成功和失败时执行的回调函数，在获取文件成功时执行的回调函数中，可以使用一个参数，代表成功获取的文件。

【示例】下面示例设计一个用于输入文件名的文本框和一个“读取文件”按钮，当用户在文件名文本框中输入文件名后，单击“读取文件”按钮，将读取该文件中的内容，并将这些内容显示在页面上的 textarea 元素中，演示效果如图 11.16 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function readFile(){      //读取文件
    window.requestFileSystem( PERSISTENT,   1024,
        function(fs){ //请求文件系统成功时所执行的回调函数
            var filename = document.getElementById("FileName").value;
            fs.root.getFile(filename,   //获取文件对象
                {create:false},
                function(fileEntry) { //获取文件对象成功时所执行的回调函数
                    fileEntry.file(    //获取文件
                        function(file) {//获取文件成功时所执行的回调函数
                            var reader = new FileReader();
                            reader.onloadend = function(e) {
                                var txtArea = document.createElement('textarea');
                                txtArea.value = this.result;
                                document.body.appendChild(txtArea);
                            };
                            reader.readAsText(file);
                        },
                        errorHandler      //获取文件失败时所执行的回调函数
                    );
                },
                errorHandler); //获取文件对象失败时所执行的回调函数
        },
        errorHandler //请求文件系统失败时所执行的回调函数
    );
}
function errorHandler(FileError) { //省略代码}
</script>
<h1>读取文件</h1>
```



```
文件名：<input type="text" id="FileName" value="test.txt"><br/> <br/>
<input type="button" value="读取文件" onclick="readFile()"> <br/>
<output id="result" ></output>
```



Note



示 例 效 果



图 11.16　读取并显示文件内容

注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。

视 频 讲 解

11.5.8　复制文件

在 FileSystem API 中，可以使用 File 对象引用磁盘文件，然后将其写入文件系统，用法如下：

```
fileWriter.write(file);
```

参数 file 表示用户磁盘上的一个文件对象。也可以为一个 Blob 对象，表示需要写入的二进制数据。在 HTML5 中，File 对象继承 Blob 对象，所以在 write()方法中可以使用 File 对象作为参数，表示使用某个文件中的原始数据进行写文件操作。

【示例】下面示例将用户磁盘上的文件复制到受浏览器沙箱保护的文件系统中。先在页面上设计一个文件控件，当用户选取磁盘上的多个文件后，将用户选取文件复制到受浏览器沙箱保护的文件系统中，复制成功后，在页面中显示所有被复制的文件名，演示效果如图 11.17 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function myfile_onchange(){ //复制文件
    var files=document.getElementById("myfile").files;
    window.requestFileSystem( PERSISTENT, 1024,
        function(fs){ //请求文件系统成功时所执行的回调函数
            for(var i = 0, file; file = files[i]; ++i){
                (function(f) {
                    fs.root.getFile(file.name, {create: true}, function(fileEntry) {
                        fileEntry.createWriter(function(fileWriter) {
                            fileWriter.onwriteend = function(e) {
                                document.getElementById("result").innerHTML+=' 复 制 文 件 名 为 ：'+f.name+'<br/>';
                            };
                            fileWriter.onerror = errorHandler
                            fileWriter.write(f);
                        }, errorHandler);
```





Note

```
                    }, errorHandler);
                })(file);
            }
        },
        errorHandler        //请求文件系统失败时所执行的回调函数
    );
}
function errorHandler(FileError) {   //省略代码   }
</script>
<h1>复制文件</h1>
<input type="file" id="myfile" onchange="myfile_onchange()" multiple /><br>
<output id="result" ></output>
```





图 11.17　复制文件内容



示 例 效 果

注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。

11.5.9　删除文件

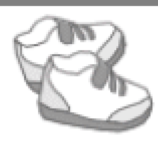


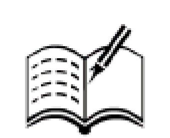
视 频 讲 解

在 FileSystem API 中，使用 FileEntry 对象的 remove()方法可以删除该文件。remove()方法包含两个参数，分别为删除文件成功和失败时执行的回调函数。

【示例】下面示例演示了如何删除指定名称的文件。在页面中设计一个文本框和一个“删除文件”按钮，用户输入文件名后，单击“删除文件”按钮，在文件系统中该文件将被删除，删除成功后在页面中显示该文件被删除的提示信息，演示效果如图 11.18 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function deleteFile(){          //删除文件
    window.requestFileSystem( PERSISTENT, 1024,
        function(fs){ //请求文件系统成功时所执行的回调函数
            var filename = document.getElementById("fileName").value;
            fs.root.getFile(//获取文件
                filename,
                { create: false },
                function(fileEntry){        //获取文件成功时所执行的回调函数
                    fileEntry.remove(
                        function() {        //删除文件成功时所执行的回调函数
                            document.getElementById("result").innerHTML =fileEntry.name+'文件被删除';
                        },
```





Note

```
                        errorHandler    //删除文件失败时所执行的回调函数
                    );
                },
                errorHandler    //获取文件失败时所执行的回调函数
            );
        },
        errorHandler //请求文件系统失败时所执行的回调函数
    );
}
function errorHandler(FileError) {    //省略代码 }
</script>
<h1>删除文件</h1>
文件名：<input type="text" id="fileName" value="test.txt"><br/><br/>
<input type="button" value="删除文件" onclick="deleteFile()"><br/>
<output id="result" ></output>
```


示 例 效 果



图 11.18　删除文件

注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。

视 频 讲 解

11.5.10　创建目录

在 FileSystem API 中，DirectoryEntry 对象表示一个目录，该对象包括如下属性：

☑ isFile：区分对象是否为文件。属性值为 true，表示对象为文件；属性值为 false，表示该对象为目录。

☑ isDirectory：区分对象是否为目录。属性值为 true，表示对象为目录；属性值为 false，表示该对象为文件。

☑ name：表示该目录的目录名。

☑ fullPath：表示该目录的完整路径。

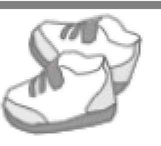
☑ filesystem：表示该目录所在的文件系统对象。

DirectoryEntry 对象还包括一些可以创建、复制或删除目录的方法。

使用 DirectoryEntry 对象的 getDirectory()方法可以在一个目录中创建或获取子目录，该方法包含 4 个参数，简单说明如下：

第 1 个参数：为一个字符串，表示需要创建或获取的子目录名。

第 2 个参数：为一个自定义对象。当创建目录时，必须将该对象的 create 属性值设定为 true；当获取目录时，必须将该对象的 create 属性值设定为 false。



第 3 个参数：为一个函数，表示获取子目录或创建子目录成功时执行的回调函数，在回调函数中可以使用一个参数，参数为一个 DirectoryEntry 对象，代表创建或获取成功的子目录。

第 4 个参数：为一个函数，表示获取子目录或创建子目录失败时执行的回调函数，参数值为一个 FileError 对象，其中存放了获取子目录或创建子目录失败时的各种错误信息。

【示例 1】下面示例演示了如何创建一个子目录。首先在页面中设计文本框，用于输入目录名称，同时添加一个“创建目录”按钮。输入目录名后，单击“创建目录”按钮，将在根目录下创建子目录，并将创建的目录信息显示在页面中，演示效果如图 11.19 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function createDirectory(){    //创建目录
    window.requestFileSystem(
        PERSISTENT,
        1024,
        function(fs){ //请求目录系统成功时所执行的回调函数
            var directoryName = document.getElementById("directoryName").value;
            fs.root.getDirectory(//创建目录
                directoryName,
                { create: true },
               function(dirEntry){    //创建目录成功时所执行的回调函数
                    var text = "目录路径：  "+dirEntry.fullPath+"<br>";
                    text += "目　录　名：  "+dirEntry.name+"<br>";
                    document.getElementById("result").innerHTML = text;
                },
                errorHandler    //创建目录失败时所执行的回调函数
            );
        },
        errorHandler  //请求文件系统失败时所执行的回调函数
    );
}
function errorHandler(FileError) { //省略代码}
</script>
<h1>创建目录</h1>
目录名：<input type="text" id="directoryName" value="test"><br/><br/>
<input type="button" value="创建目录" onclick="createDirectory()"><br/>
<output id="result" ></output>
```

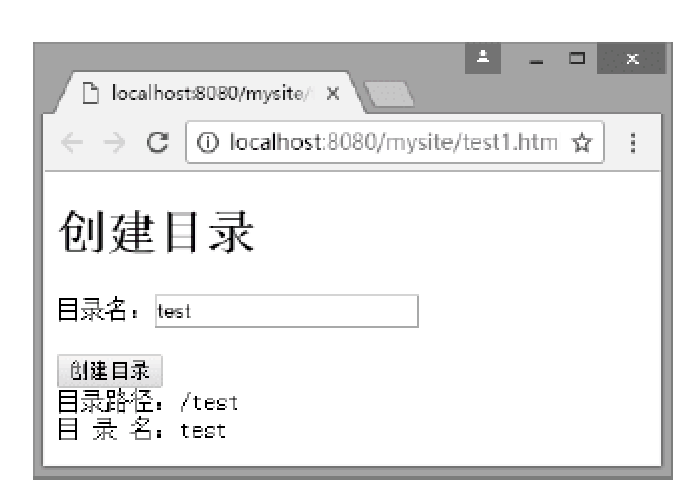


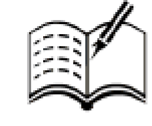
图 11.19　创建目录



示 例 效 果

在创建树形目录时，如果文件系统中不存在一个目录，直接创建该目录下的子目录时，将会抛出错误。但是有时应用程序中会执行某个操作后先创建子目录，然后创建该目录下的子目录。





Note

【示例 2】下面示例演示了如何使用递归法按正确的顺序创建子目录。在页面中显示一个“创建目录”按钮，单击该按钮后将在文件系统根目录下创建'one/two/three'这种三级目录，创建的同时在页面中按创建顺序显示被创建的每一个子目录，演示效果如图 11.20 所示。

```
<script>
var path = 'one/two/three';
function createDirectory(rootDirEntry, folders){            //创建目录
    window.requestFileSystem(
        PERSISTENT,
        1024,
        function(fs){                                        //请求文件系统成功时所执行的回调函数
            createDir(fs.root, path.split('/'));             //使用递归函数创建每一级子目录
        },
        errorHandler                                         //请求文件系统失败时所执行的回调函数
    );
}
function createDir(rootDirEntry, folders){                   //创建目录时使用的递归函数
    if (folders[0] == '.' || folders[0] == '') {             //将“/foo/.//bar”之类的目录名中的“./”或“/”剔除
        folders = folders.slice(1);
    }
    rootDirEntry.getDirectory(folders[0], {create: true},
        function(dirEntry) {                                 //创建目录成功时所执行的回调函数
            if (folders.length) {
                document.getElementById("result").innerHTML += dirEntry.name+"目录已创建<br/>";
                createDir(dirEntry, folders.slice(1));       //调用递归函数创建子目录
            }
        },
        errorHandler                                         //创建目录失败时所执行的回调函数
    );
}
function errorHandler(FileError) {                               //省略代码}
</script>
<h1>创建树形目录</h1>
<input type="button" value="创建目录" onclick="createDirectory()"><br/>
<output id="result" ></output>
```



示例效果

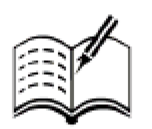


图 11.20　创建树形结构目录

注意：如果启动系统，初次测试本例，在测试本节示例之前，应先运行 11.5.1 节示例代码，以便请求访问受浏览器沙箱保护的本地文件系统，然后再运行 11.5.2 节示例代码，以便申请磁盘配额。







11.5.11　读取目录

在 FileSystem API 中，读取目录的操作步骤：

第 1 步，使用 DirectoryEntry 对象的 createReader()方法创建 DirectoryReader 对象，用法如下。

```
var dirReader=fs.root.createReader();
```

该方法不包含任何参数，返回值为创建的 DirectoryEntry 对象。

第 2 步，在创建 DirectoryEntry 对象之后，使用该对象的 readEntries()方法读取目录。该方法包含两个参数，简单说明如下：

☑ 第 1 个参数为读取目录成功时执行的回调函数。回调函数包含一个参数，代表被读取的该目录中目录及文件的集合。

☑ 第 2 个参数为读取目录失败时执行的回调函数。

第 3 步，在异步 FileSystem API 中，不能保证一次就能读取出该目录中的所有目录及文件，应该多次使用 readEntries()方法，一直到回调函数的参数集合的长度为 0 为止，表示不再读出目录或文件。

【示例】下面示例演示了如何读取目录。在页面中设计一个“读取目录”按钮，单击该按钮将读取文件系统根目录中的所有目录和文件，并将其显示在页面上，演示效果如图 11.21 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function readDirectory(){                                //读取目录
    window.requestFileSystem( PERSISTENT, 1024,
        function(fs){                                    //请求文件系统成功时所执行的回调函数
            var dirReader = fs.root.createReader();
            var entries = [];
            var readEntries = function() {               //调用 readEntries 直到读不出目录或文件
                dirReader.readEntries (
                    function(results) {                  //读取目录成功时执行的回调函数
                        if (!results.length) {
                            listResults(entries.sort());
                        }
                        else {
                            entries = entries.concat(toArray(results));
                            readEntries();
                        }
                    },
                   errorHandler                          //读取目录失败时执行的回调函数
                );
            };
            readEntries();                               //开始读取目录
        },
        errorHandler                                     //请求文件系统失败时所执行的回调函数
    );
}
function listResults(entries) {
    var type;
    entries.forEach(function(entry, i) {
        if(entry.isFile)
            type="文件："+entry.name;
```



```
            else
                type="目录："+entry.name;
            document.getElementById("result").innerHTML+=type+"<br/>";
        });
}
function toArray(list) { return Array.prototype.slice.call(list || [], 0);}
function errorHandler(FileError) { //省略代码}
</script>
<h1>读取目录</h1>
<input type="button" value="读取目录" onclick="readDirectory()"><br/>
<output id="result" ></output>
```

Note





示例效果



图 11.21　读取目录



视频讲解

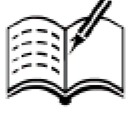
11.5.12　删除目录

在 FileSystem API 中，使用 DirectoryEntry 对象的 remove()方法可以删除该目录。该方法包含两个参数，分别为删除目录成功时执行的回调函数和删除目录失败时执行的回调函数。当删除目录时，如果该目录中含有文件或子目录，则将抛出错误。

【示例】下面示例演示了如何删除文件系统中某个目录。在页面中设计一个文本框控件和一个“删除目录”按钮，当在文本框中输入目录名后，单击“删除目录”按钮，将在文件系统中删除该目录，删除成功后在页面中显示提示信息，演示效果如图 11.22 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function deleteDirectory(){                    //删除目录
    window.requestFileSystem(
        PERSISTENT,
        1024,
        function(fs){ //请求文件系统成功时所执行的回调函数
            var directoryName = document.getElementById("directoyName").value;
            fs.root.getDirectory(                //获取目录
                directoryName,
                { create: false },
                function(dirEntry){            //获取目录成功时所执行的回调函数
                    dirEntry.removeRecursively(
                        function() {        //删除目录成功时所执行的回调函数
                            document.getElementById("result").innerHTML =dirEntry.name+'目录被删除';
                        },
                        errorHandler        //删除目录失败时所执行的回调函数
                    );
                },
```





```
                    errorHandler              //获取目录失败时所执行的回调函数
                );
            },
            errorHandler                      //请求文件系统失败时所执行的回调函数
        );
    }
    function errorHandler(FileError) {                //省略代码}
    </script>

    <h1>删除目录</h1>
    目录名：<input type="text" id="directoyName" value="test"><br/><br/>
    <input type="button" value="删除目录" onclick="deleteDirectory()"><br/>
    <output id="result" ></output>
```



图 11.22　删除目录



示例效果

提示： 当目录中含有子目录或文件时，要将该目录包括其中的子目录及文件一并删除时，可以使用 DirectoryEntry 对象的 removeRecursively()方法删除该目录。该方法包含参数及其说明与 remove()方法相同，这两个方法的不同仅在于 remove()方法只能删除空目录，而 removeRecursively()方法可以连该目录下的所有子目录及文件一并删除。

11.5.13　复制目录



视频讲解

在 FileSystem API 中，使用 FileEntry 对象或 DirectoryEntry 对象的 copyTo()方法可以将一个目录中的文件或子目录复制到另一个目录中。该方法包含四个参数：

第 1 个参数：为一个 DirectoryEntry 对象，指定将文件或目录复制到哪个目标目录中。

第 2 个参数：可选参数，为一个字符串值，用于指定复制后的文件名或目录名。

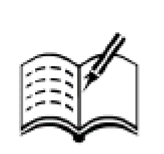
第 3 个参数：可选参数，为一个函数，代表复制成功后执行的回调函数。

第 4 个参数：可选参数，为一个函数，代表复制失败后执行的回调函数。

【示例】 下面示例使用 FileSystem API 复制文件系统中文件。页面包含三个文本框控件和一个“复制文件”按钮，其中一个文本框控件用于用户输入复制源目录，一个文本框控件用于用户输入复制的目标目录，一个文本框控件用于输入被复制的文件名，用户输入复制源目录、复制目标目录与被复制的文件名并单击“复制文件”按钮后，将被复制的文件从复制源目录复制到目标目录中，复制成功后在页面中显示提示信息，如图 11.23 所示。

```
<script>
window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;
function copyFile(){            //复制文件
    var src=document.getElementById("src").value;
    var dest=document.getElementById("dest").value;
```





Note

```
        var fileName=document.getElementById("fileName").value;
        window.requestFileSystem(window.PERSISTENT, 1024*1024, function(fs) {
            copy(fs.root, src+'/'+fileName, dest+'/');
        }, errorHandler);
    }
    function copy(cwd, src, dest) {
        cwd.getFile(src, {create:false},
        function(fileEntry) {            //获取被复制文件成功时执行的回调函数
            cwd.getDirectory(dest, {create:false},
            function(dirEntry) {         //获取复制目标目录成功时执行的回调函数
                fileEntry.copyTo(dirEntry,fileEntry.name,
                    function() {         //复制文件操作成功时执行的回调函数
                        document.getElementById("result").innerHTML ='文件复制成功';
                    },
                    errorHandler         //复制文件操作失败时执行的回调函数
                );
            },
            errorHandler);               //获取复制目标目录失败时执行的回调函数
        }, errorHandler);                //获取被复制文件失败时执行的回调函数
    }
    function errorHandler(FileError) {        //省略代码}
    </script>

    <h1>复制文件</h1>
    源  目  录：<input type="text" id="src"><br/>
    目标目录：<input type="text" id="dest"><br/>
    复制文件：<input type="text" id="fileName"><br/>
    <input type="button" value="复制文件" onclick="copyFile()">
    <output id="result" ></output>
```



示 例 效 果



图 11.23　复制目录中文件

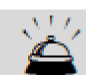


视 频 讲 解

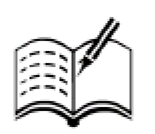
11.5.14　重命名目录

在 FileSystem API 中，使用 FileEntry 对象或 DirectoryEntry 对象的 moveTo()方法将一个目录中的文件或子目录复制到另一个目录中。该方法所用参数及其说明与 copyTo()方法完全相同。

两个方法不同点：仅在于使用 copyTo()方法时，将把指定文件或目录从复制源目录复制到目标目录中，复制后复制源目录中该文件或目录依然存在，而使用 moveTo()方法时，将把指定文件或目录从移动源目录移动到目标目录中，移动后移动源目录中该文件或目录被删除。

 提示：用户可以在 11.5.13 节示例基础上，把 copyTo()方法换为 moveTo()方法进行测试练习。





【示例】下面示例演示了如何实现文件的重命名操作。先在页面中设计三个文本框和一个“文件重命名”按钮，当在三个文本框中分别输入文件所属目录、文件名与新的文件名，单击“文件重命名”按钮后，将该文件名修改为新的文件名，修改成功后在页面上显示提示信息，如图 11.24 所示。

```
<script>
window.requestFileSystem  = window.requestFileSystem || window.webkitRequestFileSystem;
function renameFile(){                    //文件重命名
    var folder=document.getElementById("folder").value;
    var oldFileName=document.getElementById("oldFileName").value;
    var newFileName=document.getElementById("newFileName").value;
    window.requestFileSystem(window.PERSISTENT, 1024*1024, function(fs) {
        rename(fs.root, folder+'/'+oldFileName,newFileName,folder+'/');
    }, errorHandler);
}
function rename(cwd,oldFileName,newFileName,folder) {
    cwd.getFile(oldFileName, {create:false},
    function(fileEntry) {                  //获取文件成功时执行的回调函数
        cwd.getDirectory(folder, {create:false},
        function(folder) {                 //获取文件目录成功时执行的回调函数
            fileEntry.moveTo(folder,newFileName,
                function() {               //文件重命名操作成功时执行的回调函数
                    document.getElementById("result").innerHTML ='修改文件名成功，新文件名：
'+newFileName;
                },
                errorHandler               //文件重命名操作失败时执行的回调函数
            );
        },
        errorHandler);                     //获取目录失败时执行的回调函数
    }, errorHandler);                      //获取文件失败时执行的回调函数
}
function errorHandler(FileError) {              //省略代码 }
</script>

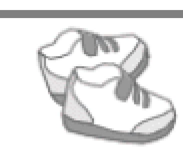
<h1>文件重命名</h1>
目 录：<input type="text" id="folder"><br/>
文件名：<input type="text" id="oldFileName"><br/>
新文件名：<input type="text" id="newFileName"><br/>
<input type="button" value="文件重命名" onclick="renameFile()"><br/>
<output id="result" ></output>
```



图 11.24　文件重命名



示例效果





视频讲解

Note

11.5.15　使用 filesystem:URL

在 FileSystem API 中，可以使用带有“filesystem:”前缀的 URL，这种 URL 通常用在页面上元素的 href 属性值或 src 属性值中。

用户可以通过 window 对象的 resolveLocalFileSystemURL()方法根据一个带有“filesystem:”前缀的 URL 获取 FileEntry 对象。该方法包含三个参数，简单说明如下：

第 1 个参数：为一个带有“filesystem:”前缀的 URL。

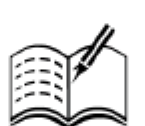
第 2 个参数：为一个函数，表示获取文件对象成功时执行的回调函数，该函数使用一个参数，表示获取到的文件对象。

第 3 个参数：为一个函数，表示获取文件对象失败时执行的回调函数，该回调函数使用一个参数，参数值为一个 FileError 对象，其中存放获取文件对象失败时的各种错误信息。

【示例】下面示例演示了“filesystem:”前缀的 URL 和 resolveLocalFileSystemURL()方法基本应用。在页面中显示一个文本框、一个“创建图片”按钮与一个“显示文件名”按钮，当输入图片文件名后，单击“创建图片”按钮，页面中显示该图片，单击“显示文件名”按钮，页面中显示该图片文件的文件名，演示效果如图 11.25 所示。

```
<script>
window.requestFileSystem  = window.requestFileSystem || window.webkitRequestFileSystem;
var fileSystemURL;
function createImg(){                    //创建图片
    window.requestFileSystem(
        PERSISTENT,
        1024,
        function(fs){                    //请求文件系统成功时所执行的回调函数
            var filename =document.getElementById("fileName").value;
            fs.root.getFile(filename,    //获取文件对象
                {create:false},
                function(fileEntry) { //获取文件成功时所执行的回调函数
                    var img = document.createElement('img');
                    fileSystemURL=fileEntry.toURL();
                    img.src = fileSystemURL;
                    document.getElementById("form1").appendChild(img);
                    document.getElementById("btnGetFile").disabled=false;
                },
                errorHandler);           //获取文件失败时所执行的回调函数
        },
        errorHandler                     //请求文件系统失败时所执行的回调函数
    );
}
function getFile(){
    window.resolveLocalFileSystemURL  =  window.resolveLocalFileSystemURL  ||window.webkitResolveLocal
FileSystemURL;
    window.resolveLocalFileSystemURL(fileSystemURL,
        function(fileEntry) {            //获取文件对象成功时执行的回调函数
            document.getElementById("result").innerHTML="文件名为:"+fileEntry.name;
        },
        errorHandler                     //获取文件对象失败时执行的回调函数
```





Note

```
        );
    }
    function errorHandler(FileError) { //省略代码}
    </script>

    <h1>使用 filesystem 前缀的 URL</h1>
    <form id="form1">
    <input type="text" id="fileName">
    <input type="button" id="btnCreateImg" value="创建图片" onclick="createImg()">
    <input type="button" id="btnGetFile" value="显示文件名" onclick="getFile()"   disabled><br/>
    </form>
    <output id="result" ></output>
```



图 11.25　显示文件



示例效果

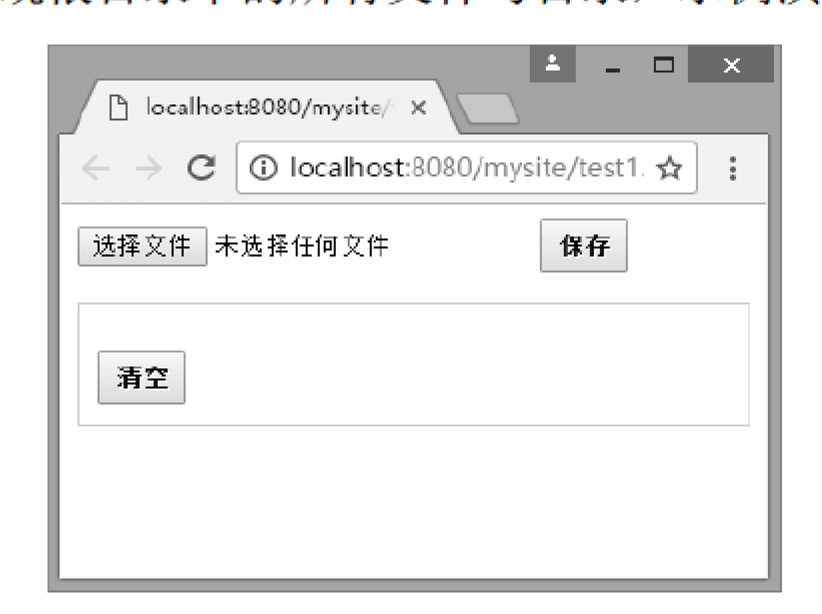
注意：在测试本节示例之前，应先运行 11.5.7 节示例代码，在文件系统中复制一个文件。

11.6　案例：设计资源管理器



视频讲解

本例设计在页面中显示一个文件控件、三个按钮。当页面打开时显示文件系统根目录下的所有文件与目录，通过文件控件可以将磁盘上一些文件复制到文件系统的根目录下，复制完成之后用户可以通过单击“保存”按钮来重新显示文件系统根目录下的所有文件与目录，单击“清空”按钮可以删除文件系统根目录下的所有文件与目录，示例演示效果如图 11.26 所示。



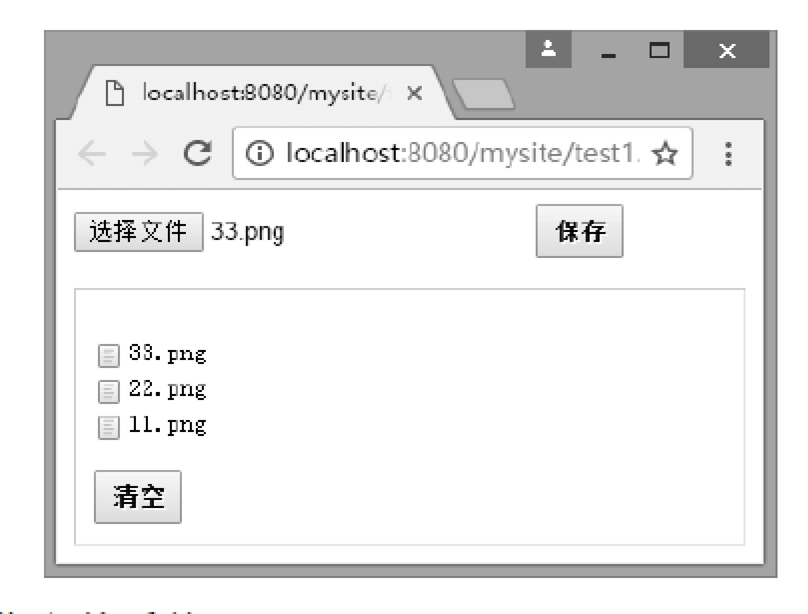


图 11.26　操作文件系统



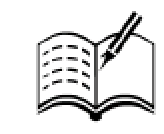
线上阅读

具体代码解析请扫码学习。





11.7 在线练习



Note

使用 File API 从 Web 网页上访问本地文件系统。



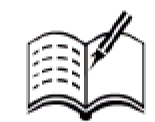
在线练习

第12章 HTML5通信

HTML5 新增多种通信技术，如跨文档消息传输功能，使用 WebSockets API 接口，通过 socket 端口传递数据的功能。使用跨文档消息传输功能，可以在不同网页文档、不同端口、不同域之间进行消息的传递。使用 WebSockets API，可以让客户端与服务器端通过 socket 端口来传递数据，这样可以实现数据推送技术，服务器端不再是被动地等待客户端发出请求，只要客户端与服务器端建立了一次连接之后，服务器端就可以在需要的时候，主动地将数据推送到客户端，直到客户端显式关闭这个连接。

【学习重点】

- 掌握怎样实现不同页面、不同端口、不同域之间的消息传递。
- 了解 WebSockets 通信技术的基本知识。
- 能够在客户端与服务器端之间建立 socket 连接，并且通过这个连接进行消息的传递。


Note

12.1 跨文档消息传递

在 Web 开发中，跨文档消息传递存在多种形式：

☑ 客户端与服务器端之间。
☑ 页面与其打开的新窗口之间。
☑ 多窗口之间。
☑ 页面与内嵌 iframe 之间。

第一种为传统用法，借助服务器技术实现，必须在同源之间通信，而后面三种可以实现跨域通信，在 JavaScript 脚本中直接完成。

12.1.1 postMessage 基础

HTML5 增加了在网页文档之间互相接收与发送信息的功能。使用这个功能，只要获取到目标网页所在窗口对象的实例，不仅同源网页之间可以互相通信，甚至可以实现跨域通信。

在 HTML5 中，跨域通信的核心是 postMessage()方法，该方法的主要功能是：向另一个地方传递数据。另一个地方可以是包含在当前页面中的 iframe 元素，或者由当前页面弹出的窗口，以及框架集中其他窗口。postMessage()方法用法如下：

```
otherWindow.postMessage(message,origin);
```

参数说明：

☑ otherWindow：表示发送消息的目标窗口对象。
☑ message：发送的消息文本，可以是数字、字符串等。HTML5 规范定义该参数可以是 JavaScript 的任意基本类型或者可复制的对象，但是部分浏览器只能处理字符串参数，考虑浏览器兼容性，可以在传递参数时，使用 JSON.stringify()方法对参数对象序列化，接收数据后再用 JSON.parse()方法把序列号字符串转换为对象。
☑ origin：字符串参数，设置目标窗口的源，格式为“协议+主机+端口号[+URL]”，URL 会被忽略，可以不写，设置该参数主要是为了安全。postMessage()只会将 message 传递给指定窗口，也可以设置该参数为“*”，这样可以传递给任意窗口，如果设置为“/”，则定义目标窗口与当前窗口同源。

目标窗口接收到消息之后，会触发 window 对象的 message 事件。这个事件以异步形式触发，因此从发送消息到接收消息，即触发目标窗口的 message 事件，可能存在延迟现象。

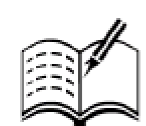
触发 message 事件后，传递给 message 处理程序的事件对象包含 3 个重要信息。

☑ data：作为 postMessage()方法第 1 个参数传入的字符串数据。
☑ origin：发送消息的文档所在域。
☑ source：发送消息的文档的 window 对象的代理。这个代理对象主要用于在发送上一条消息的窗口中调用 postMessage()方法。如果发送消息的窗口来自同一个域，该对象就是 window。

> 注意：event.source 只是 window 对象的代理，不是引用 window 对象。用户可以通过这个代理调用 postMessage()方法，但不能访问 window 对象的任何信息。

目前，Firefox 4+、Safari 4+、Chrome 8+、Opera 10+、IE 8+版本浏览器都支持这种跨文档的消息传输方式。





【示例】下面代码段定义了 HTML5 页面与内嵌框架页面之间通信的基本设计模式。

```
//发消息页面
var iframWindow = document.getElementById("myframe");
iframWindow.postMessage("发送消息", "http://www.othersite.com");

//接消息页面
var EventUtil = {//定义事件处理基本模块
    addHandler: function (element, type, handler) {          //注册事件
        if (element.addEventListener) {                      //兼容 DOM 模型
            element.addEventListener(type, handler, false);
        } else if (element.attachEvent) {                    //兼容 IE 模型
            element.attachEvent("on" + type, handler);
        } else {                                             //兼容传统模型
            element["on" + type] = handler;
        }
    }
};
EventUtil.addHandler(window, "message", function (event) {   //为 window 注册 message 事件
    //确保发送消息的域是已知的域
    if (event.origin == "http://www.mysite.com") {
        //处理接收到的数据
        processMessage(event.data);
        //可选操作：向来源窗口发送回执
        event.source.postMessage("反馈消息", event.origin);
    }
});
```

12.1.2 案例：设计简单的跨域通话



视频讲解

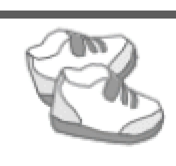
本节示例演示了如何实现跨文档消息传输。示例包含两个页面：index.html 和 called.html，其中 index.html 为主叫页面，called.html 为被叫页面。首先，主页面向内嵌 iframe 的被叫页面发送消息，被叫页面接收消息，显示在内嵌页面中，然后向主叫页面回话消息。最后，主叫页面接收消息后，将该消息显示在主页面内，整个示例演示效果如图 12.1 所示。

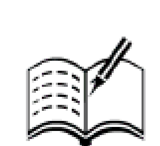


图 12.1　简单的跨文档通信演示效果

主叫页面 index.html 文档的源代码如下所示。

```
<!DOCTYPE html>
```





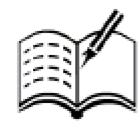
Note

```
<html>
<head>
<title>postMessage 跨域对话（主叫）</title>
<meta charset="utf-8">
<style type="text/css">
#calling { width:46%; float:left; padding:4px 12px; min-height:304px; border:solid 1px red;}
#called { width:46%; float:right; padding:4px 12px; border:solid 1px red;}
.highlight { background-color:#FFFF00;}
#iframe {width:100%; overflow:visible; min-height:300px; border:none;}
</style>
</head>
<body>
<div id="calling">
    <h1>主叫页（index.html）</h1>
    <p>对方域名：<span id="domain" class="highlight"></span></p>
    <p>对方消息：<span id="info"   class="highlight"></span></p>
</div>
<div id="called">
    <iframe id="iframe" src="http://localhost/called.html"></iframe>
</div>
<script type="text/javascript">
var EventUtil = {                                  //定义事件处理基本模块
    addHandler: function (element, type, handler) { //注册事件
        if (element.addEventListener) {            //兼容 DOM 模型
            element.addEventListener(type, handler, false);
        } else if (element.attachEvent) {          //兼容 IE 模型
            element.attachEvent("on" + type, handler);
        } else {                                   //兼容传统模型
            element["on" + type] = handler;
        }
    }
};
//窗口事件监听：监听 message
EventUtil.addHandler(window, "message", function (event) {
    document.getElementById("domain").innerHTML = event.origin;
    document.getElementById("info").innerHTML = event.data;     // event.data:发消息的内容
});
var iframe = document.getElementById("iframe");   //获取嵌入的 iframe
//当内嵌窗口加载完毕时，发送消息
EventUtil.addHandler(iframe, "load", function (event) {
    var iframeWindow = window.frames[0];
    var origin = iframe.getAttribute("src");
    iframeWindow.postMessage("called.html 页有人吗，我是主页 index.html",origin);
});
</script>
</body>
</html>
```

被叫页面 called.html 文档的源代码如下所示。

```
<!DOCTYPE html>
```






```
<html>
<head>
<title>postMessage 跨域对话（被叫）</title>
<meta charset="utf-8">
<style type="text/css">
.highlight { background-color:#FFFF00;}
</style>
</head>
<body>
<div id="call">
    <h1>被叫页（called.html）</h1>
    <p>对方域名：<span id="domain" class="highlight"></span></p>
    <p>对方消息：<span id="info"   class="highlight"></span></p>
</div>
<script type="text/javascript">
var EventUtil = {                                    //定义事件处理基本模块
    addHandler: function (element, type, handler) { //注册事件
        if (element.addEventListener) {             //兼容 DOM 模型
            element.addEventListener(type, handler, false);
        } else if (element.attachEvent) {           //兼容 IE 模型
            element.attachEvent("on" + type, handler);
        } else { //兼容传统模型
            element["on" + type] = handler;
        }
    }
};
//窗口事件监听：监听 message
EventUtil.addHandler(window, "message", function (event) {
    document.getElementById("domain").innerHTML = event.origin;
    document.getElementById("info").innerHTML = event.data;     // event.data:发消息的内容
    //收到消息后，回话
    //通过访问 message 事件对象的 source 属性，获取消息发送源的窗口对象
    //也就是能知道是谁发的消息，然后调用 postMessage()方法，给发消息者回个话
    event.source.postMessage("主页你好，我是 called.html，呼叫我有事儿吗？",event.origin);
});
</script>
</body>
</html>
```

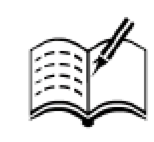
上面演示效果显示 index.html 和 called.html 同源呼叫，下面我们把 index.html 置于另一个域中，然后使用 index.html 向 called.htm 进行跨域呼叫，演示效果如图 12.2 所示。

注意：为了安全访问，在跨域传递消息过程中，应该在 message 事件中检测对方的域名，避免非法用户误入，带来安全隐患。添加的条件检测代码如下：

```
EventUtil.addHandler(window, "message", function (event) {
    // event.origin:发送消息的地址，谁发的消息就是谁的
    //接收时，可以加个判断，避免接收来源不明的 URL 发过来的数据
    if( event.origin != "http://localhost" ){
        return;
```



```
        }
        document.getElementById("domain").innerHTML = event.origin;
        document.getElementById("info").innerHTML = event.data;  // event.data:发消息的内容
});
```



Note



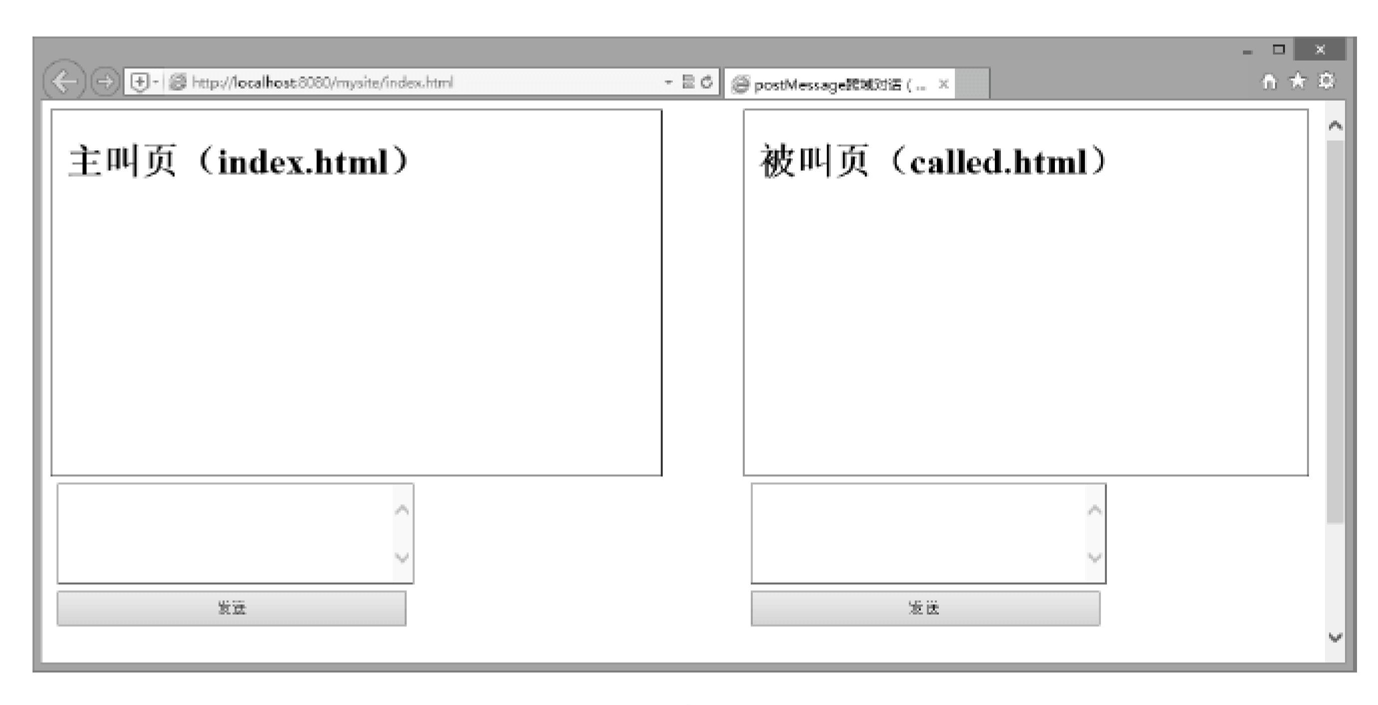
示例效果

图 12.2　简单的跨域通信演示效果

视频讲解

12.1.3　案例：设计跨域动态对话

本节示例在 12.1.2 节示例基础上，进一步改进跨域通信的互动功能。示例包含两个页面：index.html 和 called.html，其中 index.html 为主叫页面，called.html 为被叫页面。首先，主叫页面可以通过底部的文本框向被叫页面发出实时消息，被叫页面能够在底部文本框中进行动态回应，整个示例演示效果如图 12.3 所示。



（a）默认效果



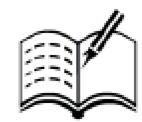


示例效果

（b）对话效果

图 12.3　跨域动态对话演示效果





主叫页面 index.html 文档的源代码如下所示。

```
<!DOCTYPE html>
<html>
<head>
<title>postMessage 跨域对话（主叫）</title>
<meta charset="utf-8">
<style type="text/css">
#calling { width: 46%; border: solid 1px red; float: left; padding: 4px 12px; min-height: 304px; }
#called { width: 46%; float: right; height: 100%; margin-top: -8px; }
.highlight { background-color: #FFFF00; }
.red { color: red; }
#iframe { width: 100%; overflow: visible; min-height: 600px; border: none; }
#caller { float: left; clear: left; }
#call_content { width: 300px; height: 80px; margin: 6px; }
#send { padding: 6px; display: block; margin-left: 6px; width: 300px; }
</style>
</head>
<body>
<div id="calling">
    <h1>主叫页（index.html）</h1>
    <div id="info"></div>
</div>
<div id="called">
    <iframe id="iframe" src="http://localhost/called.html"></iframe>
</div>
<div id="caller">
    <textarea id="call_content"></textarea>
    <button id="send" >发送</button>
</div>
<script type="text/javascript">
var EventUtil = {                                   //定义事件处理基本模块
    addHandler: function (element, type, handler) { //注册事件
        if (element.addEventListener) {             //兼容 DOM 模型
            element.addEventListener(type, handler, false);
        } else if (element.attachEvent) {           //兼容 IE 模型
            element.attachEvent("on" + type, handler);
        } else {                                    //兼容传统模型
            element["on" + type] = handler;
        }
    }
};
var info = document.getElementById("info");
var iframe = document.getElementById("iframe");
var send = document.getElementById("send");
//窗口事件监听：监听 message
EventUtil.addHandler(window, "message", function (event) {
    //监测消息来源，非法来源屏蔽
    if( event.origin != "http://localhost" ){
        return;
    }
```



Note

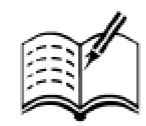
```
        info.innerHTML += '<p> 来 自 <span class="red">'+ event.origin + '</span> 的 网 友 说 ： <span class="highlight">' + event.data + '</span></p>';
    });
    EventUtil.addHandler(send, "click", function (event) {
        var iframeWindow = window.frames[0];
        var origin = iframe.getAttribute("src");
        var call_content = document.getElementById("call_content");
        if(call_content.value.length <=0) return;                  //如果文本框为空，则禁止呼叫
        iframeWindow.postMessage(call_content.value, origin);      //发出呼叫
        info.innerHTML += '<p>来自<span class="red">本页</span>的网友说：<span class="highlight">' + call_content.value + '</span></p>';
        call_content.value = "";                                   //清空文本框
    });
    </script>
    </body>
    </html>
```

被叫页面 called.html 文档的源代码如下所示。

```
    <!DOCTYPE html>
    <html>
    <head>
    <title>postMessage 跨域对话（被叫）</title>
    <meta charset="utf-8">
    <style type="text/css">
    .highlight { background-color: #FFFF00; }
    .red { color: red; }
    #call { width: 100%; border: solid 1px red; min-height: 312px; }
    #call div { padding: 4px 12px; }
    #caller { float: left; clear: left; }
    #call_content { width: 300px; height: 80px; margin: 6px; }
    #send { padding: 6px; display: block; margin-left: 6px; width: 300px; }
    </style>
    </head>
    <body>
    <div id="call">
        <div>
            <h1>被叫页（called.html）</h1>
            <div id="info"></div>
        </div>
    </div>
    <div id="caller">
        <textarea id="call_content"></textarea>
        <button id="send" >发送</button>
    </div>
    <script type="text/javascript">
    var EventUtil = {                                  //定义事件处理基本模块
        addHandler: function (element, type, handler) { //注册事件
            if (element.addEventListener) {            //兼容 DOM 模型
                element.addEventListener(type, handler, false);
            } else if (element.attachEvent) {          //兼容 IE 模型
```



```
                element.attachEvent("on" + type, handler);
            } else { //兼容传统模型
                element["on" + type] = handler;
            }
        }
    };
    var origin,source;
    var info = document.getElementById("info");
    var send = document.getElementById("send");
    EventUtil.addHandler(window, "message", function (event) {
        origin = event.origin;
        source   = event.source;
        info.innerHTML += '<p>来自<span class="red">'+ origin +'</span>的网友说：<span class="highlight">' +
event.data + '</span></p>';
    });
    EventUtil.addHandler(send, "click", function (event) {
        var call_content = document.getElementById("call_content");
        if(call_content.value.length <=0) return;
        source.postMessage(call_content.value,origin);
        info.innerHTML += '<p>来自<span class="red">本页</span>的网友说：<span class="highlight">' +
call_content.value + '</span></p>';
        call_content.value = "";
    });
    </script>
    </body>
    </html>
```



Note

12.1.4 案例：设计通道通信



视 频 讲 解

通道通信机制提供了一种在多个源之间进行通信的方法，这些源之间通过端口（port）进行通信，从一个端口中发出的数据将被另一个端口接收。

目前，IE10+、Chrome 浏览器支持通道通信机制。

> **提示**：在 HTML5 中，如果两个页面的 URL 地址位于不同的域中，或使用不同的端口号，则这两个页面属于不同的源。

使用通道通信之前，首先要创建一个 MessageChannel 对象，用法如下。

```
var me=new MessageChannel();
```

在创建 MessageChannel 对象的同时，两个端口将被同时创建，其中一个端口被本页面使用，而另一个端口将通过父页面被发送到内嵌的 iframe 元素的子页面中。

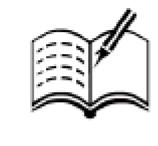
在 HTML5 通道通信中，使用的每个端口都是一个 MessagePort 对象，该对象包含 3 个方法。

☑ postMessage()：用于向通道发送消息。

☑ start()：用于激活端口，开始监听端口是否接收到消息。

☑ close()：用于关闭并停用端口。

同时每个 MessagePort 对象都有一个 message 事件，当端口接收到消息时触发该事件。与 postMessage 的 message 事件对象相似，MessagePort 的 message 事件对象包含下面几个属性：


Note

☑ data：包含任意字符串数据，由原始脚本发送。

☑ origin：一个字符串，包含原始文档的方案、域名以及端口，如：http://domain.example:80。

☑ source：原始文件的窗口的引用，它是一个 WindowProxy 对象（window 代理）。

☑ ports：一个数组，包含任何 MessagePort 对象发送消息。

【示例】下面示例演示了如何在 Web 应用中实现通道通信。为了便于理解和练习，本例使用 3 个文件夹模拟三个站点：

☑ http://localhost/site_1

☑ http://localhost/site_2

☑ http://localhost/site_3

> 注意：域名为本地一虚拟服务器，如果用户定义了虚拟目录，则需要在示例脚本中修改对应的域名。

本示例把测试主页放在 http://localhost/site_1 模拟站点（文件夹）中。在主页 index.html 中放置两个 iframe 元素，这两个 iframe 元素中放置两个子页，分别来自于 http://localhost/site_2 模拟站点和 http://localhost/site_3 模拟站点。

在浏览器中预览主页，当子页 1 被加载完毕后，向子页 2 发送消息，消息为“site_2/index.html 初始化完毕。”，当子页 2 接收到该消息后，向子页 1 返回消息，当子页 1 接收到来自子页 2 的返回消息后，将该消息在浏览器窗口中弹出显示，演示效果如图 12.4 所示。


示例效果



图 12.4　通道通信演示效果

【操作步骤】

第 1 步，在本地构建一个虚拟站点，域名为 http://localhost。

第 2 步，在站点内新建 3 个文件夹，分别命名为 site_1、site_2、site_3。

第 3 步，设计示例主页。在 site_1 目录下，新建网页保存为 index.html。

第 4 步，在 site_1/index.html 页面中嵌入两个 iframe 元素。设置 src 分别为 http://localhost/site_2/index.html 和 http://localhost/site_3/index.html。同时，使用 CSS 隐藏显示 style="display:none"。

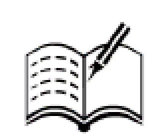
```
<iframe style="display:none" src="http://localhost/site_2/index.html"></iframe>
<iframe style="display:none" src="http://localhost/site_3/index.html"></iframe>
```

第 5 步，在首页脚本中，当页面加载完毕之后，注册一个 message 事件，监听管道消息。当接收到第一个 iframe 元素中的页面消息之后，把它转发给第二个 iframe 元素中的页面。

```
window.onload = function(){
    var iframes = window.frames;
    //对第一个 iframe 元素中的页面进行监听
```



```
        window.addEventListener('message',function(event){
            if( event.ports.length > 0 ){
                //将端口转发给第二个 iframe 元素中的页面
                iframes[1].postMessage(event.data,'http://localhost/site_3/index.html',event.ports);
            }
        },false);
    }
```



Note

第 6 步，在 site_2 目录下，新建网页保存为 index.html。在该页面脚本中，创建一个 MessageChannel 对象。

```
    var mc = new MessageChannel();
```

第 7 步，创建 MessageChannel 对象后，两个 MessagePort 对象也被同时创建，可以通过 MessageChannel 对象的 port 1 属性值与 port2 属性值来访问这两个 MessagePort 对象。

这里将 MessageChannel 对象的 port1 属性值所引用的 MessagePort 对象留在本页面，将 MessageChannel 对象的 port2 属性值所引用的 MessagePort 对象以及需要发送的消息通过父页面的 postMessage()方法发送给父页面。

```
    window.onload = function(){
        var mc = new MessageChannel();
        //向父页面发送端口及消息
        window.parent.postMessage('site_2/index.html 初始化完毕。','http://localhost/site_1/index.html',[mc.port2]);
        //定义本页面端口接收到消息时的事件处理函数
        mc.port1.addEventListener('message', function(event){
            alert( event.data );
        }, false);
        //打开本页面中的端口，开始监听
        mc.port1.start();
    }
```

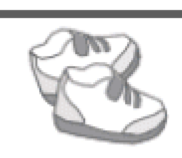
在父页面中定义接收到 MessagePort 对象及消息后，转发给其第二个 iframe 元素中的子页面，代码可以参考第 5 步说明。同时，定义 MessagePort 对象在接收到消息时所执行的事件处理函数。

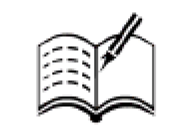
第 8 步，在 site_3 目录下，新建网页保存为 index.html。在该页面脚本中，定义当页面接收到消息及端口时，访问触发窗口对象的 message 事件的事件对象的 ports 属性值中的第一个端口，即子页 1 发送过来的端口，然后将反馈消息发回给该端口。

```
    window.onload = function(){
         //定义接收到消息时的事件处理函数
        window.addEventListener('message',function(event){
            event.ports[0].postMessage('site_3/index.html 收到消息：'+ event.data + '本页也已加载完毕。');
        },false);
    }
```

12.2 WebSockets 通信

HTML5 的 Web Sockets API 能够在 Web 应用程序中实现客户端与服务器端之间进行非 HTTP 的通信。它实现了使用 HTTP 不容易实现的服务器端的数据推送等智能通信技术，因此受到了高度关注。





提示：本节示例以 Windows 操作系统+Apache 服务器+ PHP 开发语言组合框架为基础进行演示说明。如果读者的本地系统没有搭建 PHP 虚拟服务器，建议先搭建该虚拟环境之后，再详细学习本节内容。

12.2.1　WebSocket 基础

WebSocket 是一个持久化的协议，这是相对于 HTTP 非持久化来说的。

例如，HTTP 1.0 的生命周期是以 request（请求）作为界定的，也就是一个 request、一个 response（响应）。对于 HTTP 协议来说，本次 client（客户端）与 server（服务器端）的会话到此结束；而在 HTTP 1.1 中稍微有所改进，即添加了 keep-alive，也就是在一个 HTTP 连接中可以进行多个 request 请求和多个 response 响应操作。

然而在实时通信中，HTTP 并没有多大的作用，HTTP 只能由 client 发起请求，server 才能返回信息，即 server 不能主动向 client 推送信息，无法满足实时通信的要求。而 WebSocket 可以进行持久化连接，即 client 只需要进行一次握手（类似 request），成功后即可持续进行数据通信，值得关注的是 WebSocket 能够实现 client 与 server 之间全双工通信（双向同时通信），即通信的双方可以同时发送和接收信息的信息交互方式。

图 12.5 演示了 client 和 server 之间建立 WebSocket 连接时的握手部分，这部分在 Node.js 中可以十分轻松地完成，因为 Node.js 提供的 net 模块已经对 socket 套接字做了封装处理，开发者使用的时候只需要考虑数据的交互，而不用处理连接的建立。



图 12.5　WebSocket 连接时握手示意图

client 与 server 建立 socket 时，握手的会话内容也就是 request 与 response 的过程。

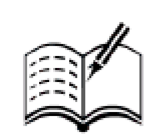
提示：socket 又称为套接字，是基于 W3C 标准开发在一个 TCP 接口中进行双向通信的技术。通常情况下，socket 用于描述 IP 地址和端口，是通信过程中的一个字符句柄。当服务器端有多个应用服务绑定一个 socket 时，通过通信中的字符句柄，实现不同端口对应不同应用服务功能。目前，大部分浏览器都支持 HTML5 的 Web Sockets API。

具体步骤：

第 1 步，client 建立 WebSocket 时，向服务器端发出请求信息：

```
GET /chat HTTP/1.1
Host: server.example.com
//告诉服务器现在发送的是 WebSocket 协议
Upgrade: websocket
Connection: Upgrade
//下面是一个 Base64 encode 的值，这个是浏览器随机生成的，用于验证服务器端返回数据是否是 WebSocket
```






```
助理
    Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
    Sec-WebSocket-Protocol: chat, superchat
    Sec-WebSocket-Version: 13
    Origin: php.cn
```

第 2 步，服务器获取到 client 请求的信息后，根据 WebSocket 协议对数据进行处理并返回，其中要对 Sec-WebSocket-Key 进行加密等操作。

```
    HTTP/1.1 101 Switching Protocols
    //依然是固定的WebSocket协议，告诉客户端即将升级的是WebSocket协议，而不是mozillasocket、lurnarsocket或者 shitsocket
    Upgrade: websocket
    Connection: Upgrade
    //下面则是经过服务器确认，并且加密过后的 Sec-WebSocket-Key，也就是 client 要求建立 WebSocket 验证的凭证
    Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
    Sec-WebSocket-Protocol: chat
```

使用 WebSockets API 可以在服务器与客户端之间建立一个非 HTTP 的双向连接。这个连接是实时的，也是永久的，除非被显式关闭。这意味着当服务器想向客户端发送数据时，可以立即将数据推送到客户端的浏览器中，无须重新建立连接。只要客户端有一个被打开的 socket（套接字），并且与服务器建立连接，服务器就可以把数据推送到这个 socket 上，服务器不再需要轮询客户端的请求，从被动转为了主动。

另外，在 WebSockets API 中。同样可以使用跨域通信技术。在使用跨域通信技术时，应该确保客户端与服务器端是互相信任的，服务器端应该判断将它的服务发送给所有客户端，还是只发送给某些受信任的客户端。

12.2.2 使用 WebSockets API

WebSocket 连接服务器和客户端，这个连接是一个实时的长连接，服务器端一旦与客户端建立了双向连接，就可以将数据推送到 Socket 中，客户端只要有一个 Socket 绑定的地址和端口与服务器建立联系，就可以接收推送来的数据。

【操作步骤】

第 1 步，创建连接。新建一个 WebSocket 对象，代码如下。

```
var host = "ws://echo.websocket.org/";
var socket=new WebSocket(host);
```

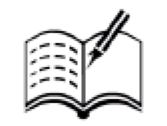
注意：WebSocket()构造函数参数为 URL，必须以“ws”或“wss”（加密通信时）字符开头，后面字符串可以使用 HTTP 地址。该地址没有使用 HTTP 协议写法，因为它的属性为 WebSocket URL。URL 必须由 4 个部分组成，分别是通信标记（ws）、主机名称（host）、端口号（port）和 WebSocket Server。

在实际应用中，socket 服务器端脚本可以是 Python、Node.js、Java、PHP。本例使用 http://www.websocket.org/网站提供的 socket 服务端，协议地址为 ws://echo.websocket.org/。这样方便初学者架设服务器测试环境，以及编写服务器脚本。



第 2 步，发送数据。当 WebSocket 对象与服务器建立连接后，使用如下代码发送数据。

```
socket.send(dataInfo);
```



Note

注意：socket 为新创建的 WebSocket 对象，send()方法中的 dataInfo 参数为字符类型，只能使用文本数据或者将 JSON 对象转换成文本内容的数据格式。

第 3 步，接收数据。通过 message 事件接收服务器传过来的数据，代码如下。

```
socket.onmessage=function(event){
    //弹出收到的信息
    alert(event.data);
    //其他代码
}
```

其中，通过回调函数中 event 对象的 data 属性来获取服务器端发送的数据内容，该内容可以是一个字符串或者 JSON 对象。

第 4 步，显示状态。通过 WebSocket 对象的 readyState 属性记录连接过程中的状态值。readyState 属性是一个连接的状态标志，用于获取 WebSocket 对象在连接、打开、变化中和关闭时的状态。该状态标志共有 4 个属性值，简单说明如表 12.1 所示。

表 12.1　readyState 属性值

属　性　值	属 性 常 量	说　　明
0	CONNECTING	连接尚未建立
1	OPEN	WebSocket 的连接已经建立
2	CLOSING	连接正在关闭
3	CLOSED	连接已经关闭或不可用

提示：WebSocket 对象在连接过程中，通过侦测 readyState 状态标志的变化，可以获取服务器端与客户端连接的状态，并将连接状态以状态码形式返回给客户端。

第 5 步，通过 open 事件监听 socket 的打开，用法如下所示。

```
webSocket.onopen = function(event){
    //开始通信时处理
}
```

第 6 步，通过 close 事件监听 socket 的关闭，用法如下所示。

```
webSocket.onclose=function(event){
    //通信结束时的处理
}
```

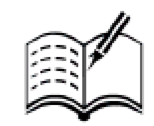
第 7 步，调用 close()方法可以关闭 socket，切断通信连接，用法如下所示。

```
webSocket.close();
```

本示例完整代码如下：

```
<html>
<head>
```



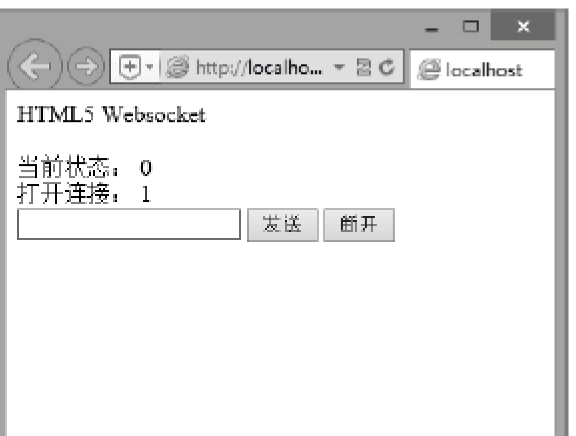


```
<script>
var socket;                                           //声明 socket
function init(){                                      //初始化
    var host = "ws://echo.websocket.org/";            //声明 host，注意是 ws 协议
    try{
        socket = new WebSocket(host);                 //新建一个 socket 对象
        log('当前状态： '+socket.readyState);          //将连接的状态信息显示在控制台
        socket.onopen     = function(msg){ log("打开连接： "+ this.readyState); };//监听连接
        socket.onmessage = function(msg){ log("接受消息： "+ msg.data); };
        //监听当接收信息时触发匿名函数
        socket.onclose    = function(msg){ log("断开接连： "+ this.readyState); }; //关闭连接
        socket.onerror    = function(msg){ log("错误信息： "+ msg.data); };         //监听错误信息
    }
    catch(ex){
        log(ex);
    }
    $("msg").focus();
}
function send(){                                      //发送信息
    var txt,msg;
    txt = $("msg");
    msg = txt.value;
    if(!msg){ alert("文本框不能够为空"); return; }
    txt.value="";
    txt.focus();
    try{ socket.send(msg); log('发送消息： '+msg); } catch(ex){ log(ex); }
 }
function quit(){                                      //关闭 socket
    log("再见");
    socket.close();
    socket=null;
}
//根据 id 获取 DOM 元素
function $(id){ return document.getElementById(id); }
//将信息显示在 id 为 info 的 div 中
function log(msg){ $("info").innerHTML+="<br>"+msg; }
//键盘事件（回车）
function onkey(event){ if(event.keyCode==13){ send(); } }
</script>
</head>
<body onload="init()">
<div>HTML5 Websocket</div>
<div id="info"></div>
<input id="msg" type="textbox" onkeypress="onkey(event)"/>
<button onclick="send()">发送</button>
<button onclick="quit()">断开</button>
</body>
</html>
```

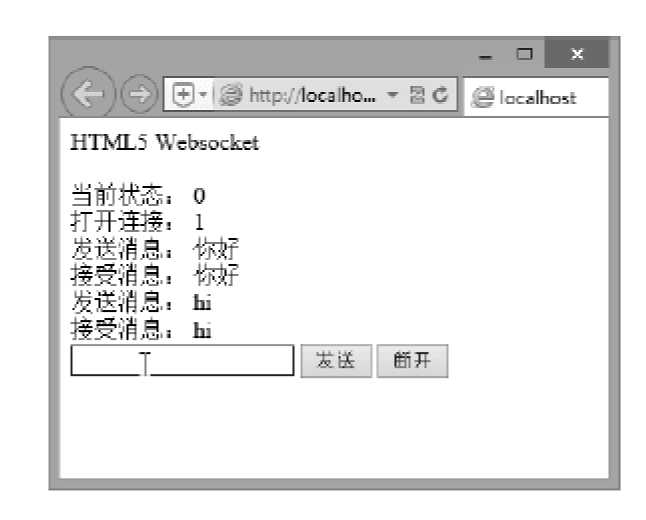
在浏览器中预览，演示效果如图 12.6 所示。



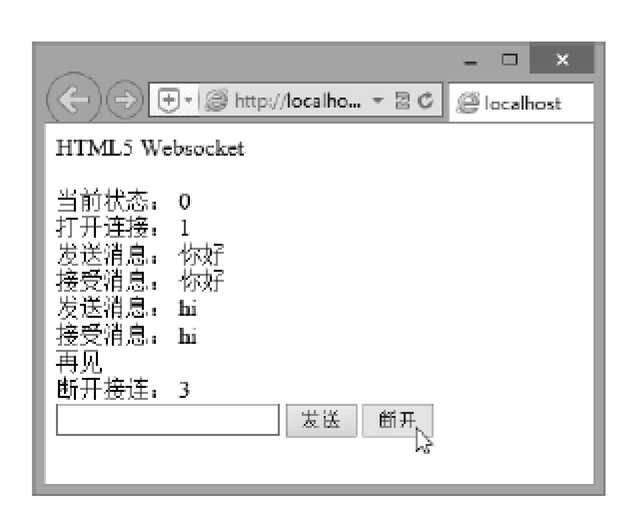
Note



（a）建立连接



（b）相互通信



（c）断开连接

图 12.6　使用 WebSocket 进行通信

在 WebSockets API 内部，通过使用 WebSocket 协议来实现多个客户端与服务器端之间的双向通信。该协议定义客户端与服务器端如何通过握手来建立通信管道，实现数据（包括原始二进制数据）的传送。

国际上标准的 WebSocket 协议为 RFC6455 协议（通过 IETF 批准），目前为止，Chrome 15+、Firefox 11+，以及 IE 10 版本的浏览器均支持该协议，包括该协议中定义的二进制数据的传送。



提示： WebSockets API 适用于当多个客户端与同一个服务器端需要实现实时通信的场景。例如，在如下所示的 Web 网站或 Web 应用程序中。

- ☑ 多人在线游戏网站。
- ☑ 聊天室。
- ☑ 实时体育或新闻评论网站。
- ☑ 实时交互用户信息的社交网站。

12.2.3　在 PHP 中建立 socket

12.2.2 节介绍了如何在前端页面中开启 WebSocket 服务，下面以 PHP 技术为基础，介绍如何在服务器端开启 WebSocket 服务。只有这样，才能够实现 client（客户端）与 server（服务器端）握手通信。

PHP 实现 WebSocket 服务主要是应用 PHP 的 socket 函数库。PHP 的 socket 函数库与 C 的 socket 函数非常类似，具体说明可以参考 PHP 参考手册。

第 1 步，在服务器中先要对已经连接的 socket 进行存储和识别。每一个 socket 代表一个用户，如何关联和查询用户信息与 socket 的对应就是一个问题，这里主要应用了文件描述符。

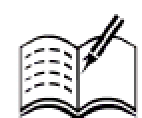
第 2 步，PHP 创建的 socket 类似于 int 值为 34 之类的资源类型，我们可以使用(int)或 intval()函数把 socket 转换为一个唯一的 ID 值，从而可以实现用一个类索引数组来存储 socket 资源和对应的用户信息：

```
$connected_sockets = array(
    (int)$socket => array(
        'resource' => $socket,
        'name' => $name,
        'ip' => $ip,
        'port' => $port,
        ...
    )
)
```



第 3 步，创建服务器端 socket 的代码如下：

```
//创建一个 TCP socket：此函数的可选值在 PHP 参考手册中有详细说明，这里不再展开
$this->master = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
//配置参数：设置 IP 和端口重用，在重启服务器后能重新使用此端口
socket_set_option($this->master, SOL_SOCKET, SO_REUSEADDR, 1);
//绑定通道：将 IP 和端口绑定在服务器 socket 上
socket_bind($this->master, $host, $port);
//监听通道：listen 函数使主动连接套接口变为被连接套接口，使得此 socket 能被其他 socket 访问，从而实现服务器功能。后面的参数则是自定义的待处理 socket 的最大数目，并发高的情况下，这个值可以设置大一点，虽然它也受系统环境的约束。
socket_listen($this->master, self::LISTEN_SOCKET_NUM);
```



这样就得到一个服务器 socket，当有客户端连接到此 socket 上时，它将改变状态为可读。

第 4 步，完成通道连接之后，下面是服务器的处理逻辑。

这里着重讲解一下 socket_select()函数的用法：

```
int socket_select(array &$read, array &$write, array &$except, int $tv_sec[, int $tv_usec = 0 ] )
```

socket_select()函数使用传统的 select 模型，可读、可写、异常的 socket 会被分别放入$read、$write、$except 数组中，然后返回状态改变的 socket 的数目，如果发生了错误，函数将会返回 false。

注意： 最后两个时间参数只有单位不同，可以搭配使用，用来表示 socket_select 阻塞的时长。为 0 时此函数立即返回，可以用于轮询机制；为 null 时，函数会一直阻塞下去，这里可设置 $tv_sec 参数为 null，让它一直阻塞，直到有可操作的 socket 返回。

下面是服务器的主要逻辑：

```
$write = $except = NULL;
$sockets = array_column($this->sockets, 'resource'); //获取到全部的 socket 资源
$read_num = socket_select($sockets, $write, $except, NULL);
foreach ($sockets as $socket) {
    //如果可读的是服务器 socket， 则处理连接逻辑
    if ($socket == $this->master) {
        socket_accept($this->master);
        // socket_accept()接收请求的连接，即一个客户端 socket，错误时返回 false
        self::connect($client);
        continue;
    //如果可读的是其他已连接 socket，则读取其数据，并处理应答逻辑
    } else {
        //函数 socket_recv()从 socket 中接收长度为 len 字节的数据，并保存在$buffer 中
        $bytes = @socket_recv($socket, $buffer, 2048, 0);
        if ($bytes < 9) {
            //当客户端忽然中断时，服务器会接收到一个 8 字节长度的消息（由于其数据帧机制，8 字节的消息表示它是客户端异常中断消息）
            //服务器处理下线逻辑，并将其封装为消息广播出去
            $recv_msg = $this->disconnect($socket);
        } else {
            //如果此客户端还未握手，执行握手逻辑
            if (!$this->sockets[(int)$socket]['handshake']) {
                self::handShake($socket, $buffer);
                continue;
```



Note

```
                } else {
                    $recv_msg = self::parse($buffer);
                }
            }
            //广播消息
            $this->broadcast($msg);
        }
    }
```

上面代码只是服务器处理消息的基础代码，日志记录和异常处理都略过了，而且还有些数据帧解析和封装的方法，在此不再展开。

12.2.4 WebSockets API 开发框架

WebSockets API 的使用为 Web 应用程序的搭建提供了一种新的架构。自 HTML5 开始，在一个 Web 应用程序中，当服务器端不会同时处理过多来自客户端的请求时，仍然可以使用传统的 Web 应用程序架构。例如，LAMP（Linux 操作系统+Apache 服务器+ MySQL 数据库+PHP 或 Perl 或 Python）架构。

当服务器端需要同时处理大量来自客户端的请求，并且需要确保服务器端只需花费较少的性能成本来处理这些请求时，可以使用这种新型的使用 WebSockets API 的应用程序架构，因为这种应用程序架构通常使用“非阻塞型 IO”技术。

目前为止，能够实现这种新型的使用 WebSockets API 的应用程序架构的开发框架包括：

☑ 使用 Node.js 开发语言
  - Socket.IO
  - WebSocket-Node
  - ws
☑ 使用 Java 开发语言
  - Jetty
☑ 使用 Ruby 开发语言
  - EventMachine
☑ 使用 Python 开发语言
  - Pywebsocket
  - Tornado
☑ 使用 Erlang 开发语言
  - Shirasu
☑ 使用 C++开发语言
  - .Libwebsockets
☑ 使用.NET 开发语言
  - Superwebsocket

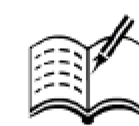


视频讲解

12.2.5 案例：设计简单的“呼-应”通信

本节通过一个简单的示例演示如何使用 WebSockets 让客户端与服务器端握手连接，然后进行简单的呼叫和应答通信。





【操作步骤】

第 1 步，新建客户端页面，保存为 client.html。

第 2 步，在页面中设计一个简单的交互表单。其中<textarea id="data">用于接收用户输入，单击<button id="send">按钮，可以把用户输入的信息传递给服务器，服务器接收到信息之后，响应信息并显示在<div id="message">容器中。

```
<div id="action">
    <textarea id="data"></textarea>
    <button id="send">发送信息</button>
</div>
<div id="message"> </div>
```

第 3 步，设计 JavaScript 脚本，建立与服务器端的连接，并通过 open、message、error 事件处理函数跟踪连接状态。

```
<script>
var message = document.getElementById('message');
var socket = new WebSocket('ws://127.0.0.1:8008');
socket.onopen = function(event) {
    message.innerHTML = '<p>连接成功！ </p>';
}
socket.onmessage = function(event) {
    message.innerHTML =  "<p>响应信息："+ event.data  +"</p>";
}
socket.onerror = function() {
    message.innerHTML = '<p>连接失败！ </p>';
}
</script>
```

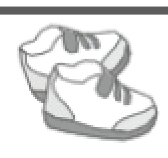
第 4 步，获取用户输入的信息，并把它发送给服务器。

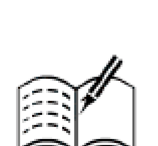
```
var send = document.getElementById('send');
send.addEventListener('click', function() {     //设计单击按钮提交信息
    var content = document.getElementById('data').value;
    if(content.length <= 0){                     //验证信息
        alert('消息不能为空！ ');
        return false;
    }
    socket.send(content);                        //发送信息
});
```

第 5 步，服务器端应用程序开发。新建 PHP 文件，保存为 server.php，与 client.html 同置于 PHP 站点根目录下。

第 6 步，为了方便操作，定义 WebSocket 类，结构代码如下。

```
<?php
//定义 WebSocket 类
class WebSocket {
    private $socket;//socket 的连接池，即 client 连接进来的 socket 标志
    private $accept;//不同状态的 socket 管理
    private $isHand = array();// 判断是否握手
```





Note

```
    //在构造函数中创建 socket 连接
    public function __construct($host, $port, $max) { }
    //对创建的 socket 循环进行监听，处理数据
    public function start() { }
    //首次与客户端握手
    public function dohandshake($sock, $data, $key) {   }
    //关闭一个客户端连接
    public function close($sock) {   }
    //解码过程
    public function decode($buffer) {   }
    //编码过程
    public function encode($buffer) {   }
}
?>
```

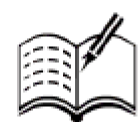
第 7 步，在构造函数中创建 socket 连接。

```
public function __construct($host, $port, $max) {
    //创建服务端的 socket 套接流，net 协议为 IPv4，protocol 协议为 TCP
    $this->socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
    socket_set_option($this->socket, SOL_SOCKET, SO_REUSEADDR, TRUE);
    //绑定接收的套接流主机和端口，与客户端相对应
    socket_bind($this->socket, $host, $port);
    //监听套接流
    socket_listen($this->socket, $max);
}
```

第 8 步，监听并接收数据。

```
    public function start() {
        while(true) {//死循环，让服务器无限获取客户端传过来的信息
            $cycle = $this->accept;
            $cycle[] = $this->socket;
            socket_select($cycle, $write, $except, null); //这个函数同时接收多个连接
            foreach($cycle as $sock) {
                if($sock === $this->socket) {//如果有新的 client 连接进来
                    $client = socket_accept($this->socket); //接收客户端传过来的信息
                    $this->accept[] = $client; //将新连接进来的 socket 存进连接池
                    $key = array_keys($this->accept); //返回包含数组中所有键名的新数组
                    $key = end($key); //输出数组中最后一个元素的值
                    $this->isHand[$key] = false; //标志该 socket 资源没有完成握手
                } else {
                    //读取该 socket 的信息，
                    //注意：第二个参数是引用传参，即接收数据
                    //第三个参数是接收数据的长度
                    $length = socket_recv($sock, $buffer, 204800, 0);
                    //根据 socket 在 accept 池里面查找相应的键 ID
                    $key = array_search($sock, $this->accept);
                    //如果接收的信息长度小于 7，则该 client 的 socket 为断开连接
                    if($length < 7) {
                        $this->close($sock); //给该 client 的 socket 进行断开操作
                        continue;
```





```
                    }
                    if(!$this->isHand[$key]) {//判断该 socket 是否已经握手
                        //如果没有握手，则进行握手处理
                        $this->dohandshake($sock, $buffer, $key);
                    } else {//向该 client 发送信息，对接收到的信息进行 uncode 处理
                        //先解码，再编码
                        $data = $this->decode($buffer);
                        $data = $this->encode($data);
                        //判断断开连接（断开连接时数据长度小于 10）
                        //如果不为空，则进行消息推送操作
                        if(strlen($data) > 0) {
                            foreach($this->accept as $client) {
                                //向 socket_accept 套接流写入信息，也就是反馈信息给 socket_bind()所绑定
的主机客户端，socket_write 的作用是向 socket_create 的套接流写入信息，或者向 socket_accept 的套接流写入信
息
                                socket_write($client, $data, strlen($data));
                            }
                        }
                    }
                }
            }
        }
    }
```

第 9 步，定义 dohandshake()函数，建立与客户端的第一次握手连接。

```
    //首次与客户端握手
    public function dohandshake($sock, $data, $key) {
        //截取 Sec-WebSocket-Key 的值并加密，其中$key 后面的一部分 258EAFA5-E914-47DA-95CA-
C5AB0DC85B11 字符串应该是固定的
        if (preg_match("/Sec-WebSocket-Key: (.*)\r\n/", $data, $match)) {
            $response = base64_encode(sha1($match[1] . '258EAFA5-E914-47DA-95CA-C5AB0DC85B11',
true));
            $upgrade  = "HTTP/1.1 101 Switching Protocol\r\n" .
                    "Upgrade: websocket\r\n" .
                    "Connection: Upgrade\r\n" .
                    "Sec-WebSocket-Accept: " . $response . "\r\n\r\n";
            socket_write($sock, $upgrade, strlen($upgrade));
            $this->isHand[$key] = true;
        }
    }
```

关于解码和编码函数，这里不再详细说明，读者可以参考本节示例源代码。

第 10 步，实例化 WebSocket 类型，并调用 start()方法开通 WebSocket 服务。

```
//127.0.0.1 是在本地主机测试，如果有多台计算机，可以写 IP 地址
$webSocket = new WebSocket('127.0.0.1', 8008, 10000);
$webSocket->start();
```

第 11 步，在浏览器中先运行 server.php，启动 WebSocket 服务器，此时页面没有任何信息，浏览器一致等待客户端页面的连接请求，如图 12.7 所示。

第 12 步，在浏览器中先运行 client.html，可以看到客户端与服务器端握手成功，如图 12.8 所示。



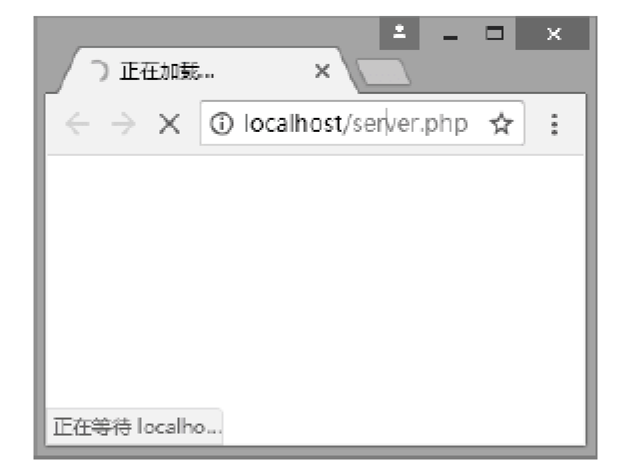




图 12.7　启动 WebSocket 服务器

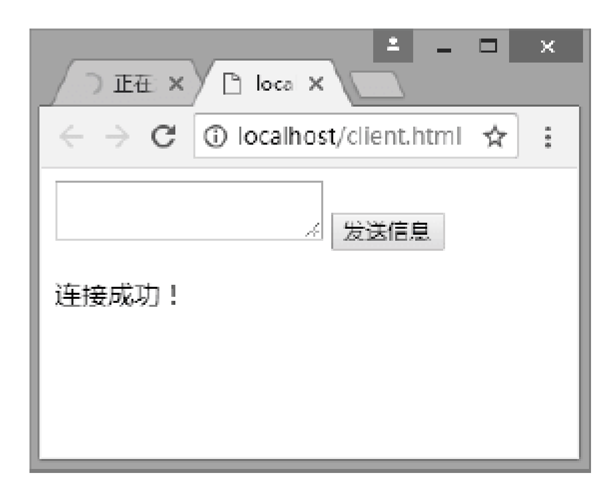


图 12.8　握手成功

第 13 步，在 client.html 页面中向服务器发送一条信息，则服务器会通过 WebSocket 通道返回一条响应信息，如图 12.9 所示。

提示： 直接在浏览器中运行 WebSocket 服务器，PHP 的配置参数（php.ini）有个时间限制，如下所示，也可以通过 “new WebSocket('127.0.0.1', 8008, 10000);” 中第 3 个参数控制轮询时长，超出这个时限，就会显示如图 12.10 所示提示错误。

default_socket_timeout = 60

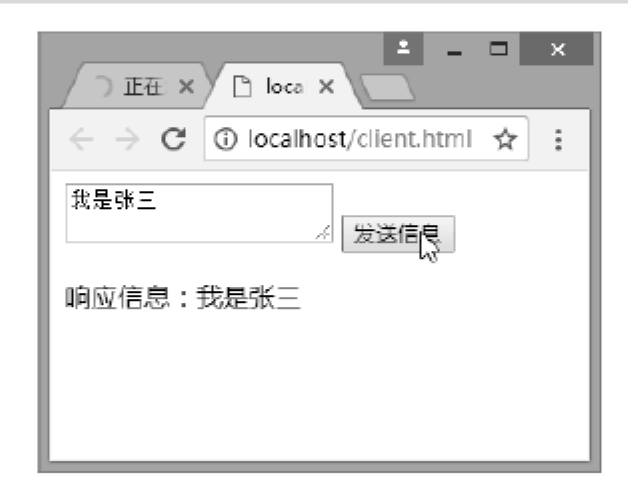


图 12.9　相互通信

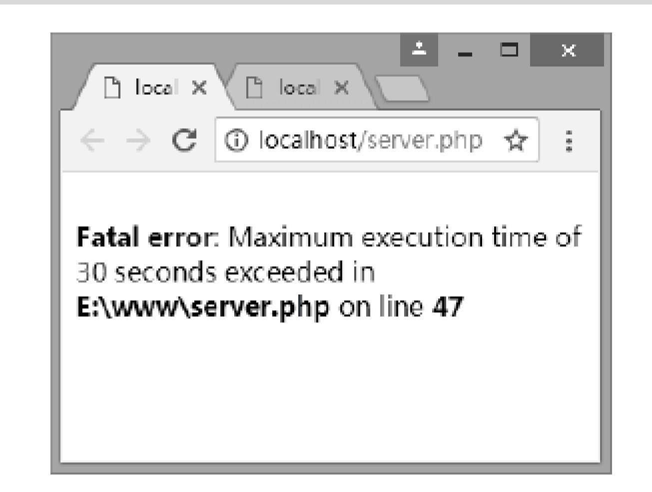


图 12.10　超出时限提示错误

【拓展】

用户也可以通过命令行运行 WebSocket 服务，实现长连接。具体操作步骤如下：

第 1 步，在“运行”对话框中启动命令行工具，如图 12.11 所示。

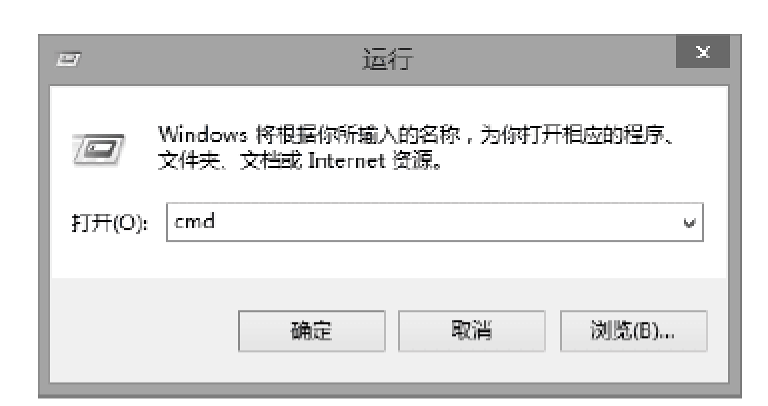


图 12.11　打开命令行

第 2 步，在命令行中输入“php　E:\www\server.php”，然后按 Enter 键运行 WebSocket 服务器应用程序即可，如图 12.12 所示。

第 3 步，只要不关闭命令行窗口，用户可以随时在客户端使用 WebSocket 与服务器端进行通信，或者服务器主动向用户推送信息。

提示： 在命令行中能够正确使用 php 命令时，应该在 Windows 环境中设置好环境变量，如图 12.13 所示。方法：在“控制面板\系统\高级系统设置”的“高级”选项中找到“环境变量”按钮，单击即可打开下图对话框，设置好 php.exe 在本地系统中所在的路径即可。





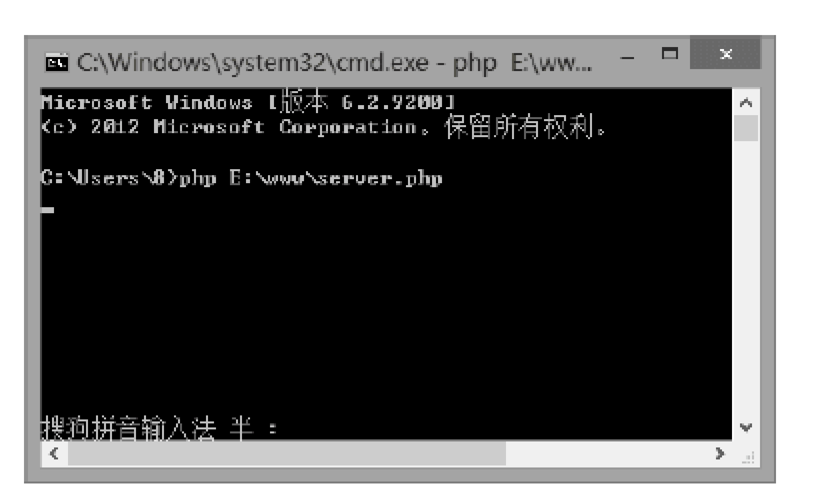



图 12.12　运行 WebSocket 服务

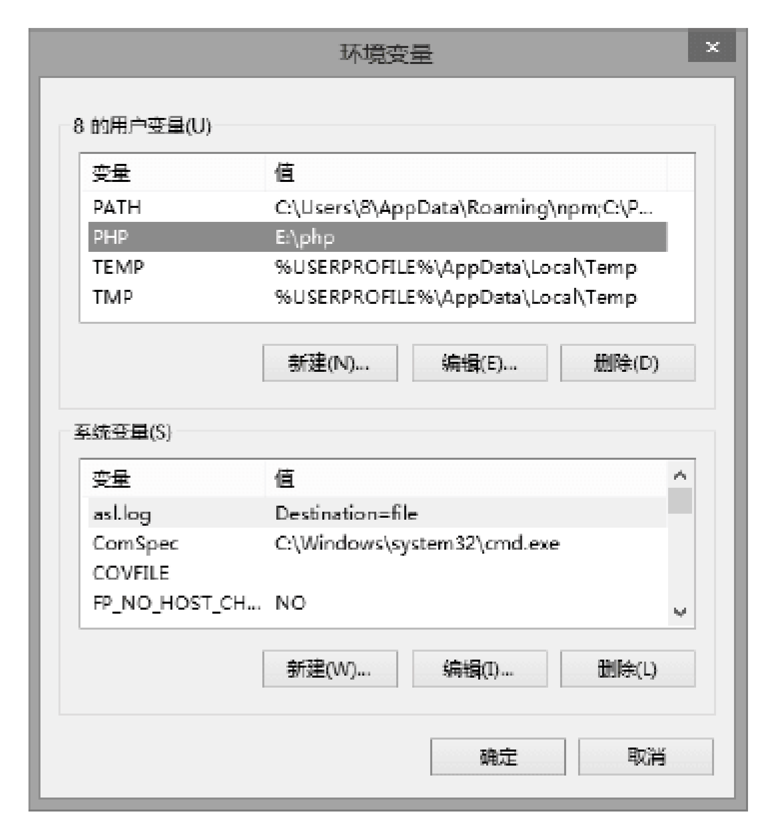



图 12.13　设置 PHP 环境变量





12.2.6　案例：发送 JSON 对象

12.2.5 节示例介绍了如何使用 WebSockets API 发送文本数据，本节示例将演示如何使用 JSON 对象来发送一切 JavaScript 中的对象。使用 JSON 对象的关键是使用它的两个方法：JSON.parse 和 JSON.stringify，其中 JSON.stringify()方法可以把 JavaScript 对象转换成文本数据，JSON.parse()方法可以将文本数据转换为 JavaScript 对象。

本示例在 12.2.5 节示例基础上进行设计，这里仅简单修改部分代码。看一下怎么使用 JSON 对象来发送和接收 JavaScript 对象。

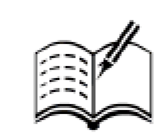
【操作步骤】

第 1 步，复制 12.2.5 节 client.html 文件，在按钮单击事件处理函数中生成一个 JSON 对象，向服务器传递两个数据：一个是随机数，一个是用户自己输入的字符串。

```
send.addEventListener('click', function() {
    var content = data.value;
    var message = {
        "randoms" :   Math.random(),  //生成随机数
        "content" : content  //用户输入的任意字符串
    }
    var json = JSON.stringify(message);//把 JSON 对象转换为字符串
    socket.send(json);                  //发送字符串信息
});
```

第 2 步，在 onmessage 事件处理函数中接收字符串信息，然后把它转换为 JSON 对象，然后稍加处理并显示在页面中。

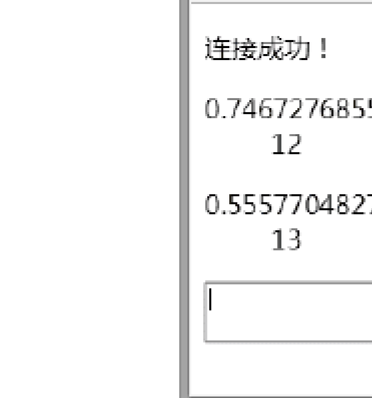




Note

```
socket.onmessage = function(event) {
    var dl = document.createElement('dl');
    var jsonData = JSON.parse(event.data);   //接收推送信息，并转换为 JSON 对象
    dl.innerHTML =   "<dt>"+jsonData.randoms +"<dt><dd><span></span>"+jsonData.content+"</dd>";
    message.appendChild(dl);
    message.scrollTop = message.scrollHeight;
}
```

第 3 步，复制 12.2.5 节 server.php 文件，保持源代码不变。然后，按 12.2.5 节操作步骤，在浏览器中进行测试，则演示效果如图 12.14 所示。



示 例 效 果



图 12.14　解析 JSON 对象并显示键值



视 频 讲 解

12.2.7　案例：使用 Workerman 框架通信

直接使用 PHP 编写 WebSockets 应用服务比较烦琐，对于初学者来说是一个挑战。本节介绍如何使用 Workerman 框架简化 WebSockets 应用开发。

注意：类似 Workerman 的框架比较多，如 Node.js、Netty、Undertow、Jetty、Spray-websocket、Vert.x、Grizzly 等，本节介绍的 Workerman 框架比较简单、实用。

Workerman 是一个高性能的 PHP socket 服务器框架，其目标是让 PHP 开发者更容易地开发出基于 socket 的高性能的应用服务，而不用去了解 PHP socket 以及 PHP 多进程细节。

【操作步骤】

第 1 步，访问 https://github.com/walkor/workerman，下载 Workerman 框架。

第 2 步，把压缩文件 Workerman-master.zip 解压到本地站点根目录下，解压并重命名文件夹为 Workerman。

第 3 步，新建 server.php 文件，输入下面代码，启用 Workerman。

```
<?php
//导入库文件
use Workerman\Worker;
require_once 'Workerman/Autoloader.php';
//创建一个 Worker 监听 2346 端口，使用 websocket 协议通信
$ws_worker = new Worker("websocket://127.0.0.1:8008");
//启动 4 个进程对外提供服务
$ws_worker->count = 4;
//当收到客户端发来的数据后返回响应信息 $data 给客户端
$ws_worker->onMessage = function($connection, $data){

    $connection->send( $data);    //向客户端发响应信息 $data
```

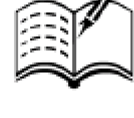


```
};
//运行
Worker::runAll();
?>
```

第 4 步，模仿 12.2.5 节示例操作，在命令行中输入下面命令启动服务，如图 12.15 所示。

```
php E:\www\test\server.php
```

注意：具体路径要结合本地系统的物理路径而定。



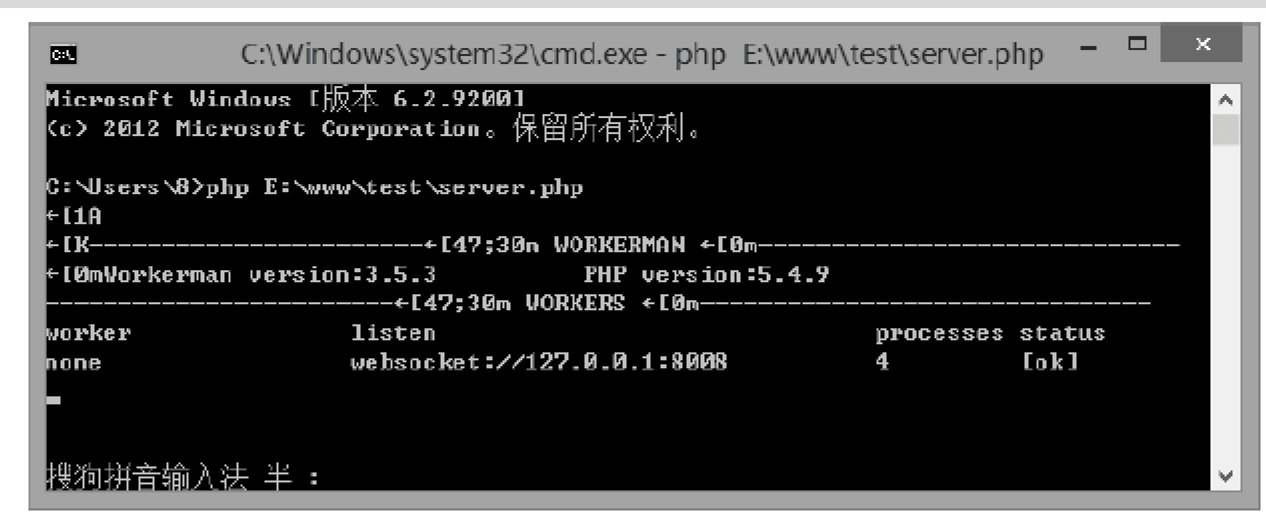


图 12.15　启动服务

第 5 步，显示上图提示信息，说明 WebSockets 应用服务启动成功。然后复制 12.2.5 节示例中的 client.html，在浏览器中预览，则可以进行握手通信了，演示效果如图 12.16 所示。

注意：Workerman 服务不能够直接在浏览器中启动，否则会显示如图 12.17 所示的提示信息。

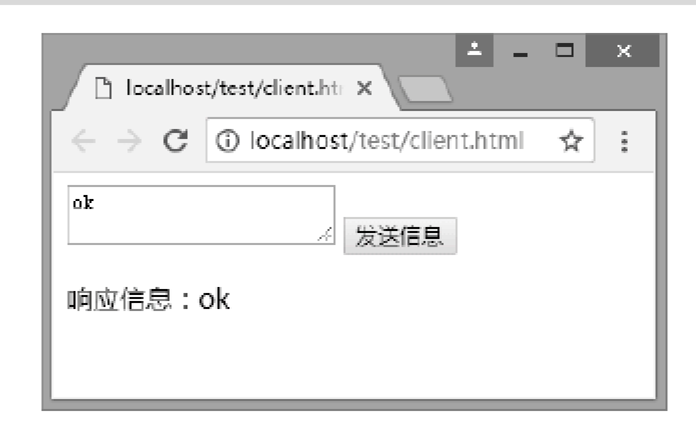


图 12.16　握手通信



图 12.17　提示错误



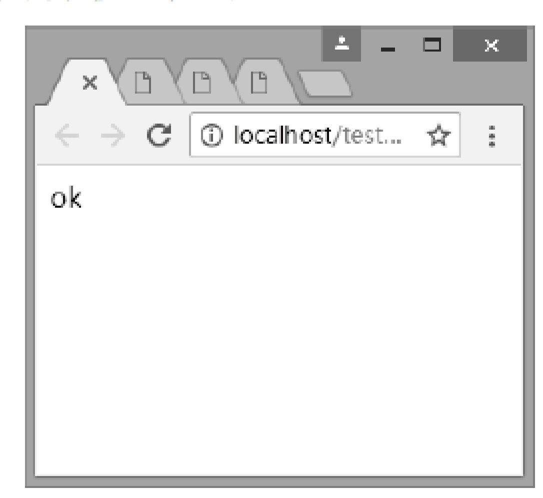
示例效果



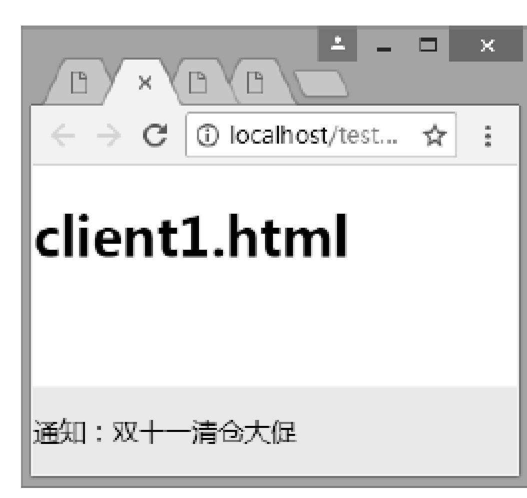
视频讲解

12.2.8　案例：推送信息

本节示例模拟微信推送功能，为特定会员主动推送优惠广告信息。在浏览器中运行 push.php，向客户端 uid 为 2 的会员推送信息，则可以看到 client1.html、client2.html 显示通知信息，如图 12.18 所示，而 client3.html 没有收到通知。



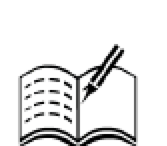
（a）推送成功



（b）client1 收到信息

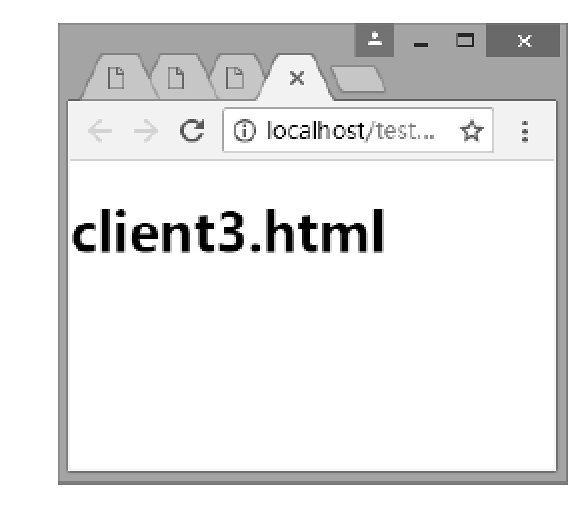
图 12.18　向特定会员推送信息








（c）client2 收到信息



（d）client3 没有收到信息

线上阅读

图 12.18　向特定会员推送信息（续）

具体操作步骤请扫码学习。

12.3　在线练习

使用 WebSocket 进行通信。



在线练习

第13章 WebRTC视频直播

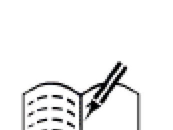
传统的 Web 通信，主要是借助服务器搭桥来实现不同用户之间的联系。随着点对点技术的出现，在两个浏览器之间直接实现通信成为可能，特别是在音频聊天、视频聊天、大数据传输、多人游戏场景中，这种技术优势尽显。WebRTC 是众多点对点技术的一个典范，本章将重点介绍 WebRTC 技术的使用。

【学习重点】

- 了解 WebRTC 的作用及其组成部分。
- 使用 getUserMedia 方法访问本地音频、视频输入设备。
- 掌握怎样实现 WebRTC 通信。



13.1 WebRTC基础



众所周知，浏览器本身不支持相互之间直接建立信道进行通信，都是通过服务器进行中转。例如，现在有两个客户端：甲和乙，它们想要通信，首先需要甲和服务器、乙和服务器之间建立信道。甲给乙发送消息时，甲先将消息发送到服务器上，服务器对甲的消息进行中转，发送到乙处，反过来也是一样。这样甲与乙之间的一次消息要通过两段信道，通信的效率同时受制于这两段信道的带宽。同时这样的信道并不适合数据流的传输，如何建立浏览器之间的点对点传输，一直困扰着开发者，WebRTC应运而生。

WebRTC表示网页实时通信（Web Real-Time Communication），是一个支持网页浏览器进行实时语音对话或视频对话的技术。它是一个开源项目，旨在使浏览器能为实时通信（RTC）提供简单的JavaScript接口。简单来说，就是让浏览器提供JavaScript的即时通信接口。这个接口所创立的信道并不像WebSocket一样，打通一个浏览器与WebSocket服务器之间的通信，而是通过一系列的信令，建立一个浏览器与另一个浏览器之间（peer-to-peer）的信道，这个信道可以发送任何数据，而不需要经过服务器，并且WebRTC通过实现MediaStream，通过浏览器调用设备的摄像头、麦克风，使得浏览器之间可以传递音频和视频。

目前，Chrome 26+、Firefox 24+、Opera 18+版本的浏览器均支持WebRTC的实现。在Chrome和Opera浏览器中将RTCPeerConnection命名为webkitRTCPeerConnection，在Firefox浏览器中将RTCPeerConnection命名为mozRTCPeerConnection，不过随着WebRTC标准的稳定，各个浏览器前缀将会被移除。

13.2 案例实战

WebRTC实现了三个API，简单说明如下：

☑ MediaStream：能够通过设备的摄像头和麦克风获得视频、音频的同步流。

☑ RTCPeerConnection：用于构建点对点之间稳定、高效的流传输的组件。

☑ RTCDataChannel：在浏览器间（点对点）建立一个高吞吐量、低延时的信道，用于传输任意数据。

13.2.1 访问本地设备

MediaStream API为WebRTC提供了从设备的摄像头、麦克风获取视频、音频流数据的功能。用户可以通过调用navigator.getUserMedia()访问本地设备，该方法包含三个参数：

☑ 约束对象（constraints object）。

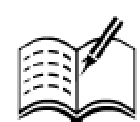
☑ 调用成功的回调函数，如果调用成功，传递给它一个流对象。

☑ 调用失败的回调函数，如果调用失败，传递给它一个错误对象。

提示：由于浏览器实现的不同，经常会在标准版本的方法前面加上前缀，一个兼容版本代码如下：

```
javacriptvar getUserMedia = (navigator.getUserMedia ||
```






```
                                  navigator.webkitGetUserMedia ||
                                  navigator.mozGetUserMedia ||
                                  navigator.msGetUserMedia);
```

【示例】下面示例演示如何调用 getUserMedia()方法访问本地摄像头。

```
<video id="myVideo" width="400" height="300" autoplay></video>
<script>
navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || window.navigator.mozGetUserMedia;
window.URL = window.URL || window.webkitURL;
var video = document.getElementById('myVideo');
navigator.getUserMedia({video:true, audio:false},
function(stream) {
    video.src = window.URL.createObjectURL(stream);
},
function(err) {
    console.log(err);
 });
</script>
```

在 Chrome 浏览器中打开页面，浏览器将首先询问用户是否允许脚本访问本地摄像头，如图 13.1 所示。当用户单击“允许”按钮后，浏览器中会显示从用户本地摄像头中捕捉到的影像，如图 13.2 所示。

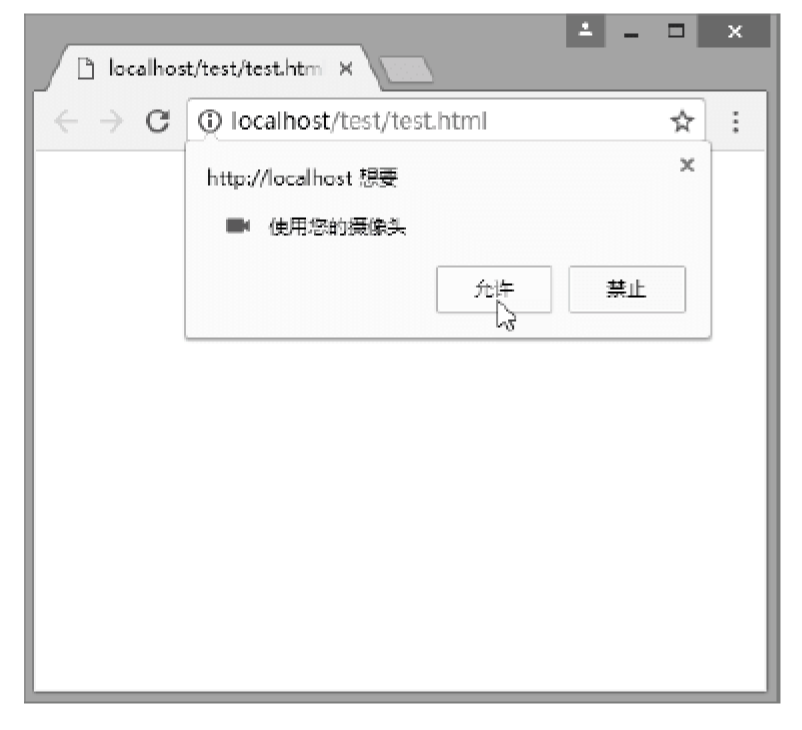




图 13.1 询问权限

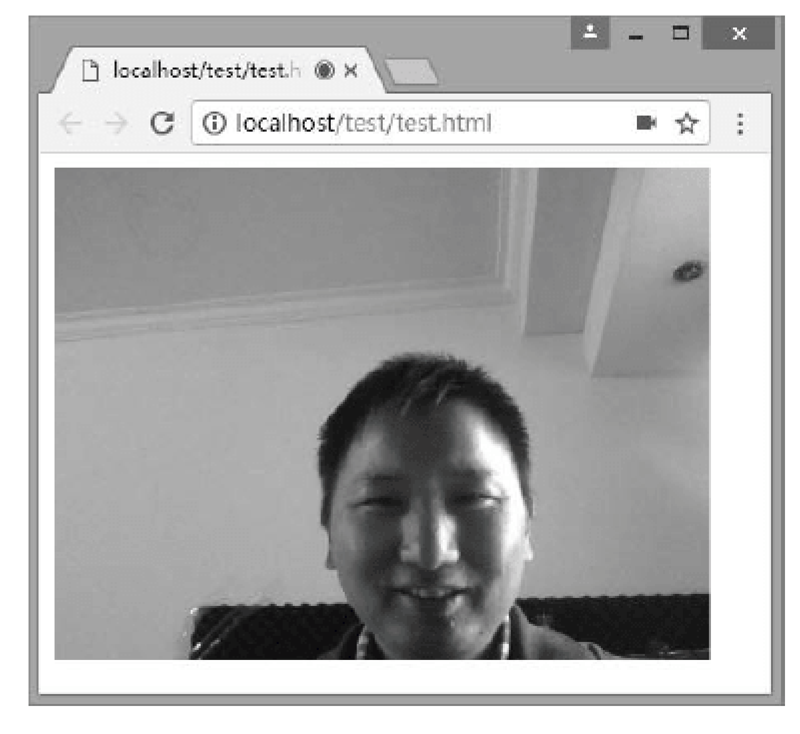


图 13.2 捕捉摄像头视频流



示例效果

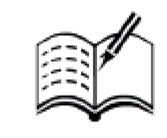
注意：HTML 文件要放在服务器上，否则会得到一个 NavigatorUserMediaError 的错误，显示 PermissionDeniedError。也可以在命令行中使用 cd 命令，具体方法：进入 HTML 文件所在目录，python -m SimpleHTTPServer（需要安装 python）。然后，在浏览器地址栏中输入 http://localhost:8000/test.html，按 Enter 键即可。

在 getUserMedia()方法中，第一个参数值为一个约束对象，该对象包含一个 video 属性和一个 audio 属性，属性值均为布尔类型。当 video 属性值为 true 时，表示捕捉视频信息；当 video 属性值为 false 时，表示不捕捉视频信息；当 audio 属性值为 true 时，表示捕捉音频信息；当 audio 属性值为 false 时，表示不捕捉音频信息。浏览器弹出要求用户给予权限的请求时，也会根据约束对象的不同而有所改变。注意，在一个浏览器标签中设置的 getUserMedia()约束将影响之后打开的所有标签中的约束。

第二个参数值为访问本地设备成功时所执行的回调函数，该回调函数具有一个参数，参数值为一



个 MediaStream 对象，当浏览器执行 getUserMedia()方法时将自动创建该对象。该对象代表同步媒体数据流。例如，一个来自于摄像头、麦克风输入设备的同步媒体数据流往往是来自于视频轨道和音频轨道的同步数据。



Note

每一个 MediaStream 对象都拥有一个字符串类型的 ID 属性，如“e1c55526-a70b-4d46-b5c1-dd19f9dc6beb”，以标识每一个同步媒体数据流。该对象的 getAudioTracks()方法或 getVideoTracks()方法将返回一个 MediaStreamTrack 对象的数组。

MediaStreamTrack 对象表示一个视频轨道或一个音频轨道，每一个 MediaStreamTrack 对象包含两个属性：

- ☑ kind：字符串类型，标识轨道种类，如“video”或“audio”。
- ☑ label：字符串类型，标识音频通道或视频通道，如“HP Truevision HD (04f2:b2f8)”。

getUserMedia 方法中第三个参数值为访问本地设备失败时所执行的回调函数，该回调函数具有一个参数，参数值为一个 error 对象，代表浏览器抛出的错误对象。

上面示例结合 HTML5 的 video 元素。window 对象的 URL.createObjectURL()方法允许将一个 MediaStream 对象转换为一个 Blob URL 值，以便将其设置为一个 video 元素的属性，这样可以通过 video.src 把视频流显示在网页中。

注意：同时为 video 元素设置 autoplay 属性，如果不使用该属性，则 video 元素将停留在所获取的第一帧画面位置。

【拓展】

约束对象可以被设置在 getUserMedia()和 RTCPeerConnection 的 addStream()方法中，这个约束对象是 WebRTC 用来指定接收什么样的流的，其中可以定义如下属性：

- ☑ video：是否接收视频流。
- ☑ audio：是否接收音频流。
- ☑ MinWidth：视频流的最小宽度。
- ☑ MaxWidth：视频流的最大宽度。
- ☑ MinHeight：视频流的最小高度。
- ☑ MaxHiehgt：视频流的最大高度。
- ☑ MinAspectRatio：视频流的最小宽高比。
- ☑ MaxAspectRatio：视频流的最大宽高比。
- ☑ MinFramerate：视频流的最小帧速率。
- ☑ MaxFramerate：视频流的最大帧速率。

13.2.2 视频截图

本例使用 Canvas API 的图形上下文对象的 drawImage()方法将 video 元素中的某一帧视频画面输出到 canvas 元素中，实现实时截图功能。

示例主要代码如下所示：

```
<video id="myVideo" width="400" height="300" autoplay></video>
<button id="btn">视频截图</button>
<img src="" id="img" ></img>
<canvas width="800" height="600" style="display:none;" id="canvas" ></canvas>
```



Note

```
<script >
navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || window.navigator.mozGetUserMedia;
window.URL = window.URL || window.webkitURL;
var video = document.getElementById('myVideo');
var btn = document.getElementById('btn');
btn.addEventListener('click', snapshot, false);
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
var localMediaStream = null;
navigator.getUserMedia({video:true, audio:false},
function(stream) {
    video.src = window.URL.createObjectURL(stream);
    localMediaStream = stream;
},
function(err) {
    console.log(err);
});
function snapshot() {
    if (localMediaStream) {
        ctx.drawImage(video, 0, 0);
        document.getElementById('img').src = canvas.toDataURL('image/png');
    }
}
</script>
```

在浏览器中访问页面，单击“视频截图”按钮，页面中的 img 元素将显示鼠标单击时 video 元素中显示的影像，如图 13.3 所示。

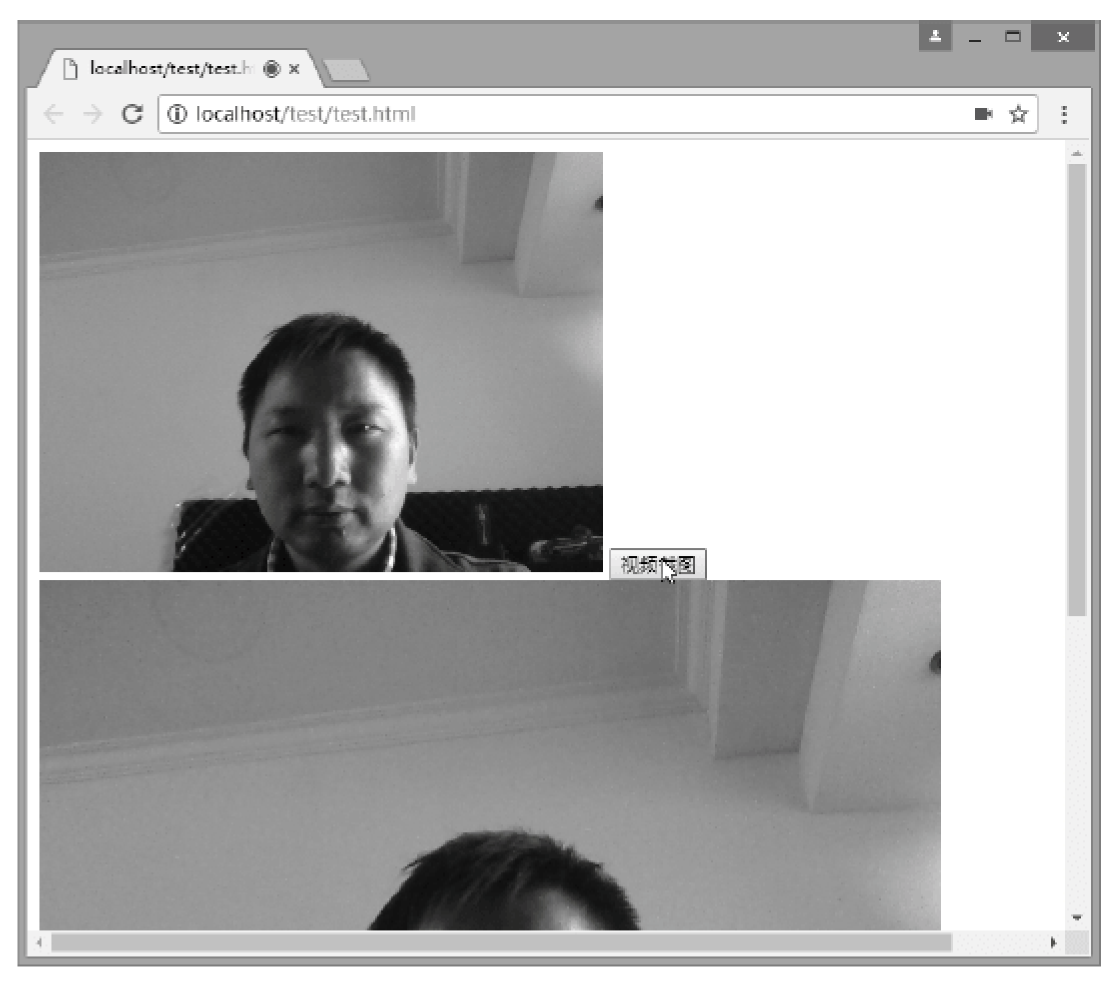


图 13.3　视频截图

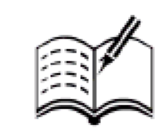


示 例 效 果

13.2.3　视频对话基础

WebRTC 使用 RTCPeerConnection 在浏览器之间传递流数据，这个流数据通道是点对点的，不需要经过服务器进行中转。但是这并不意味着服务器无用，因为还需要服务器传递信令（signaling）来





Note

建立这个信道。WebRTC 没有定义用于建立信道的信令的协议：信令并不是 RTCPeerConnection API 的一部分。

由于没有定义具体的信令的协议，用户可以选择任意方式（如 Ajax、WebSocket），采用任意的协议（如 SIP、XMPP）来传递信令，建立信道。

需要信令来交换的信息包括三种：

☑ session 信息：用来初始化通信，还有报错信息等。
☑ 网络配置：如 IP 地址和端口等。
☑ 媒体适配：发送方和接收方的浏览器能够接收什么样的编码器和分辨率。

这些信息的交换应该在点对点的流传输之前就全部完成。

通过服务器建立信道，WebRTC 需要服务器对其进行四方面的功能支持：

☑ 用户发现以及通信。
☑ 信令传输。
☑ NAT/防火墙穿越。
☑ 如果点对点通信建立失败，可以作为中转服务器。

建立点对点信道的主要问题就是 NAT/防火墙穿越技术。在 RTCPeeConnection 中，使用 ICE 框架可以保证 RTCPeerConnection 能实现 NAT 穿越。

浏览器兼容方法：

```
var PeerConnection = (window.PeerConnection ||
                      window.webkitPeerConnection00 ||
                      window.webkitRTCPeerConnection ||
                      window.mo 上 zRTCPeerConnection);
```

13.2.4 视频对话实现

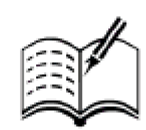
为了在浏览器与浏览器之间进行 P2P 通信，必须首先知道对方的 IP 地址及动态分配的 UDP 端口号。因此在建立 P2P 通信之前，首先需要使用 WebRTC 交换下面这些信息。

☑ SDP：一种会话描述协议，它以字符串的形式显示如下浏览器信息。
  ➢ 在浏览器与浏览器之间所进行的会话中将要使用的媒体种类（音频、视频）、媒体格式（codec）。
  ➢ 通信双方所使用的 IP 地址和端口号。
  ➢ P2P 数据传输协议，在 WebRTC 中为 Secure RTP。
  ➢ 通信时所使用的带宽。
  ➢ 会话属性，如名称、标识符、激活时间等。
☑ ICE：一种以 UDP 为基础的请求/回答模式的多媒体会话，用于实现 NAT 穿越的协议。它以清单的形式描述 P2P 通信时，可以使用如下通信途径。
  ➢ 使用 P2P 进行直接通信。
  ➢ 使用 STUN（为了穿越 NAT 而进行端口映射）实现突破 NAT 网关的 P2P 通信。
  ➢ 使用 TURN 中继服务器进行突破防火墙的中继通信。

ICE 协议在网络上通过最短途径（网络负荷最小的途径）选择被发现的候选者，并按优先级依序列举这些候选者。

在 P2P 通信之前，首先需要双方交换 SDP 和 ICE 信息，这个过程称为“信令”。事实上，WebRTC



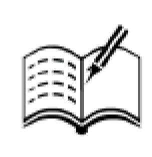


中并不规定信令方式，用户可以自由选择。下面示例演示了一种原始的方式，通过用户复制/粘贴的方式为两个浏览器传递信令。

示例主要代码如下所示：

```
<body>
<button type="button" onclick="startVideo();">捕获视频</button>
<button type="button" onclick="stopVideo();">停止捕获</button>
<button type="button" onclick="connect();">连接</button>
<button type="button" onclick="hangUp();">挂断</button><hr />
<div>
    <video id="local-video" autoplay></video>
    <h3>本人</h3>
</div>
<div>
    <video id="remote-video" autoplay></video>
    <h3>对方</h3>
</div><hr />
<p>发送的 SDP 字符串:<br /><textarea id="text-for-send-sdp" disabled="1"></textarea></p>
<p>接收的 SDP 字符串:<br /><textarea id="text-for-receive-sdp"></textarea><br />
    <button type="button" onclick="onSDP();">验证 SDP</button></p>
<hr />
<p>发送的 ICE 字符串:<br /><textarea id="text-for-send-ice" disabled="1"></textarea></p>
<p> 接收的 ICE 字符串:<br /><textarea id="text-for-receive-ice"></textarea><br />
    <button type="button" onclick="onICE();">验证 ICE</button></p>
<script>
var localVideo = document.getElementById('local-video');
var remoteVideo = document.getElementById('remote-video');
var localStream = null;
var peerConnection = null;
var peerStarted = false;
var mediaConstraints = {'mandatory': {'OfferToReceiveAudio':false, 'OfferToReceiveVideo':true }};
var textForSendSDP = document.getElementById('text-for-send-sdp');
var textForSendICE = document.getElementById('text-for-send-ice');
var textToReceiveSDP = document.getElementById('text-for-receive-sdp');
var textToReceiveICE = document.getElementById('text-for-receive-ice');
var iceSeparator = '------ ICE 候选者  -------';
var CR = String.fromCharCode(13);
//交换信息
function onSDP() {
    var text = textToReceiveSDP.value;
    var evt = JSON.parse(text);
    if (peerConnection) {
        onAnswer(evt);
    }else {
        onOffer(evt);
    }
    textToReceiveSDP.value ="";
}
function onICE() {
    var text = textToReceiveICE.value;
```

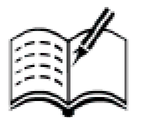




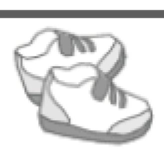
Note

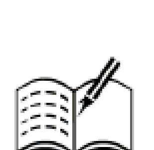
```
            var arr = text.split(iceSeparator);
            for (var i = 1, len = arr.length; i < len; i++) {
                var evt = JSON.parse(arr[i]);
                onCandidate(evt);
            }
            textToReceiveICE.value ="";
        }
        function onOffer(evt) {
            console.log("接收到 offer...")
            console.log(evt);
            setOffer(evt);
            sendAnswer(evt);
        }
        function onAnswer(evt) {
            console.log("接收到 Answer...")
            console.log(evt);
            setAnswer(evt);
        }
        function onCandidate(evt) {
            var candidate = new RTCIceCandidate({sdpMLineIndex:evt.sdpMLineIndex, sdpMid:evt.sdpMid,
candidate:evt.candidate});
            console.log("接收到 Candidate...")
            console.log(candidate);
            peerConnection.addIceCandidate(candidate);
        }
        function sendSDP(sdp) {
            var text = JSON.stringify(sdp);
            textForSendSDP.value = text;
        }
        function sendCandidate(candidate) {
            var text = JSON.stringify(candidate);
            console.log(text);
            textForSendICE.value = (textForSendICE.value + CR + iceSeparator + CR + text + CR);
            textForSendICE.scrollTop = textForSendICE.scrollHeight;
        }
        //视频处理
        function startVideo() {
            navigator.webkitGetUserMedia({video: true, audio: true},
            function (stream) { // success
                localStream = stream;
                localVideo.src = window.URL.createObjectURL(stream);
                localVideo.play();
                localVideo.volume = 0;
            },
            function (error) { // error
                console.error('发生了一个错误: [错误代码： ' + error.code + ']');
                return;
            });
        }
        function stopVideo() {
```





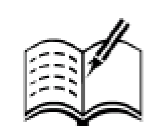
```
        localVideo.src = "";
        localStream.stop();
    }
    //处理连接
    function prepareNewConnection() {
        var pc_config = {"iceServers":[]};
        var peer = null;
        try {
            peer = new webkitRTCPeerConnection(pc_config);
        } catch (e) {
            console.log("建立连接失败，错误： " + e.message);
        }
        //发送所有 ICE 候选者给对方
        peer.onicecandidate = function (evt) {
            if (evt.candidate) {
                console.log(evt.candidate);
                sendCandidate({type: "candidate",
                              sdpMLineIndex: evt.candidate.sdpMLineIndex,
                              sdpMid: evt.candidate.sdpMid,
                              candidate: evt.candidate.candidate});
            }
        };
        console.log('添加本地视频流...');
        peer.addStream(localStream);
        peer.addEventListener("addstream", onRemoteStreamAdded, false);
        peer.addEventListener("removestream", onRemoteStreamRemoved, false);
        //当接收到远程视频流时，使用本地 video 元素进行显示
        function onRemoteStreamAdded(event) {
            console.log("Added remote stream");
            remoteVideo.src = window.URL.createObjectURL(event.stream);
        }
        //当远程结束通信时，取消本地 video 元素中的显示
        function onRemoteStreamRemoved(event) {
            console.log("移除远程视频流");
            remoteVideo.src = "";
        }
        return peer;
    }
    function sendOffer() {
        peerConnection = prepareNewConnection();
        peerConnection.createOffer(function (sessionDescription) { //成功时调用的回调函数
            peerConnection.setLocalDescription(sessionDescription);
            console.log("发送: SDP");
            console.log(sessionDescription);
            sendSDP(sessionDescription);
        }, function (err) { //失败时调用的回调函数
            console.log("创建 Offer 失败");
        }, mediaConstraints);
    }
    function setOffer(evt) {
```






```
        if (peerConnection) {
            console.error('peerConnection 已存在!');
        }
        peerConnection = prepareNewConnection();
        peerConnection.setRemoteDescription(new RTCSessionDescription(evt));
    }
    function sendAnswer(evt) {
        console.log('发送 Answer,创建远程会话描述...' );
        if (! peerConnection) {
            console.error('peerConnection 不存在!');
            return;
        }
        peerConnection.createAnswer(function (sessionDescription) { //成功时调用的回调函数
            peerConnection.setLocalDescription(sessionDescription);
            console.log("发送: SDP");
            console.log(sessionDescription);
            sendSDP(sessionDescription);
        }, function () { //失败时调用的回调函数
            console.log("创建 Answer 失败");
        }, mediaConstraints);
    }
    function setAnswer(evt) {
        if (! peerConnection) {
            console.error('peerConnection 不存在!');
            return;
        }
        peerConnection.setRemoteDescription(new RTCSessionDescription(evt));
    }
    //处理用户 UI 事件
    //开始建立连接
    function connect() {
        if (!peerStarted && localStream) {
            sendOffer();
            peerStarted = true;
        } else {
            alert("请首先捕获本地视频数据.");
        }
    }
    //停止连接
    function hangUp() {
        console.log("挂断.");
        stop();
    }
    function stop() {
        peerConnection.close();
        peerConnection = null;
        peerStarted = false;
    }
    </script>
    </body>
```





注意： 本例需要在支持 SDP 协议和 ICE 协议的 Web 服务器中运行，如 Apache 服务器。

上机测试步骤如下：

第 1 步，在 Chrome 或 Opera 浏览器中预览本例文件，分别在两个浏览器窗口或标签页中访问页面地址。

第 2 步，分别单击“捕获视频”按钮，然后单击“允许”按钮允许浏览器访问本地摄像头和麦克风设备，页面中将显示从本地摄像头实时捕获的影像，如图 13.4 所示。

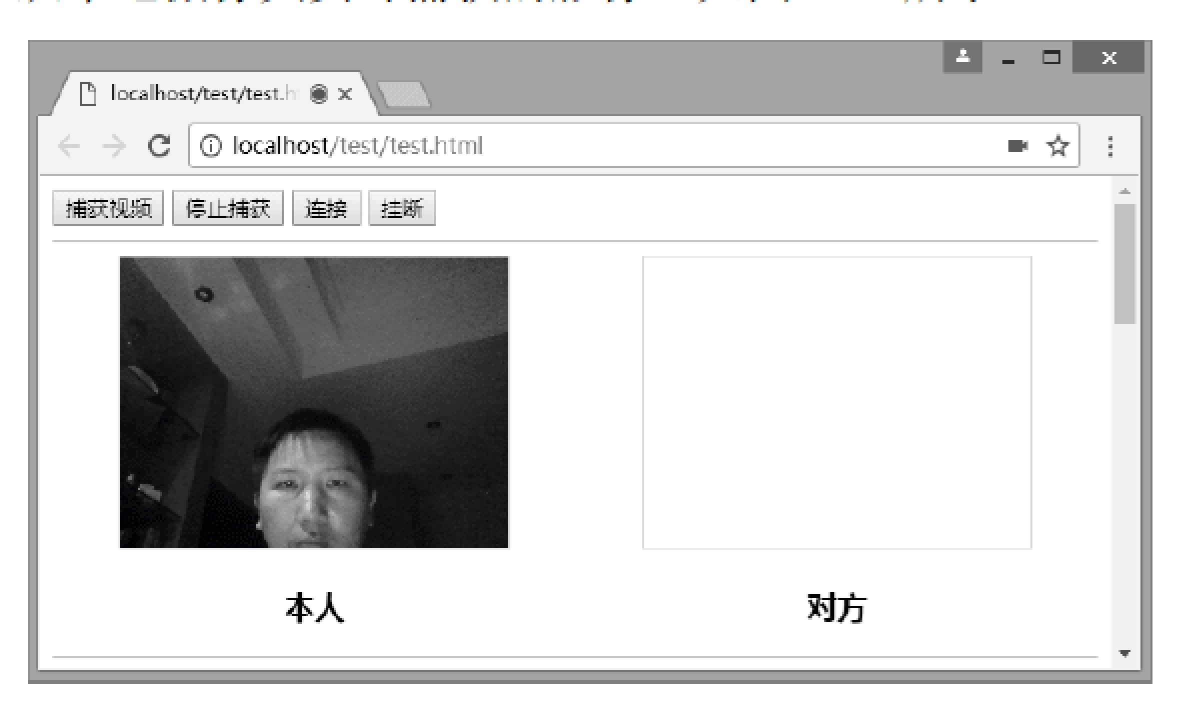


图 13.4　捕获本地视频流

第 3 步，在第一个浏览器标签页中单击“连接”按钮，这时在“发送的 SDP 字符串”文本框中自动生成一段 SDP 字符串，如图 13.5 所示。



图 13.5　生成的 SDP 字符串

第 4 步，复制第一个浏览器标签页中生成的 SDP 字符串，即“发送的 SDP 字符串:”文本框中的全部文本。

第 5 步，切换到第二个浏览器标签页中，在“接收的 SDP 字符串:”文本框中粘贴第一个浏览器标签页生成的 SDP 字符串，如图 13.6 所示。

第 6 步，单击“验证 SDP”按钮，第二个浏览器标签页内“发送的 SDP 字符串:”文本框显示远程的 SDP 描述字符串，如图 13.7 所示。

第 7 步，将第二个浏览器标签页内“发送的 SDP 字符串:”文本框中的 SDP 描述字符串全部复制到第一个浏览器标签页内“接收的 SDP 字符串:”文本框中。然后单击“验证 SDP”按钮，完成 SDP 信息交换过程。



Note



图 13.6　传递 SDP 字符串



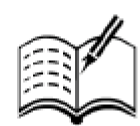
图 13.7　远程 SDP 字符串

第 8 步，将第一个浏览器标签页内“发送的 ICE 字符串:”文本框中的全部文本复制到第二个浏览器标签页内“接收的 ICE 字符串:”文本框中，如图 13.8 所示。



图 13.8　复制本地 ICE 字符串





Note

第 9 步，在第二个浏览器标签页内单击“验证 ICE”按钮，完成 ICE 信息交换验证，如图 13.9 所示。



图 13.9　验证本地 ICE 字符串

第 10 步，如果需要，还需要将第二个浏览器标签页内“发送的 ICE 字符串:”文本框中的全部文本复制到第一个浏览器标签页内的“接收的 ICE 字符串:”文本框中，单击第一个浏览器标签页内“验证 ICE”按钮。

第 11 步，现在两边窗口同时显示摄像头中捕捉到的影像，通信正式开始。第一个浏览器标签中的显示效果如图 13.10 所示。



图 13.10　显示效果



示 例 效 果

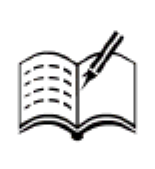
13.2.5　SDP 交换

本节重点分析 13.2.4 节示例的 SDP 交换过程。

第 1 步，当单击“连接”按钮之后，将调用 connect()函数，开始建立连接，代码如下所示：

```
function connect() {
    if (!peerStarted && localStream) {
        sendOffer();
        peerStarted = true;  //标识变量，标识开始视频对话
    } else {
        alert("请首先捕获本地视频数据.");
    }
}
```





Note

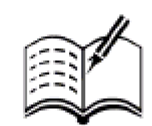
第 2 步，在 connect()函数中调用 sendOffer()函数。sendOffer()函数的代码如下所示：

```
function sendOffer() {
    peerConnection = prepareNewConnection();
    peerConnection.createOffer(function (sessionDescription) { //成功时调用的回调函数
        peerConnection.setLocalDescription(sessionDescription);
        console.log("发送: SDP");
        console.log(sessionDescription);
        sendSDP(sessionDescription);
    }, function (err) {  //失败时调用的回调函数
        console.log("创建 Offer 失败");
    }, mediaConstraints);
}
```

第 3 步，在 sendOffer()函数中调用 prepareNewConnection()函数，准备处理连接。prepareNewConnection()函数的代码如下所示。

```
function prepareNewConnection() {
    var pc_config = {"iceServers":[]};
    var peer = null;
    try {
        peer = new webkitRTCPeerConnection(pc_config);
    } catch (e) {
        console.log("建立连接失败，错误：" + e.message);
    }
     //发送所有 ICE 候选者给对方
    peer.onicecandidate = function (evt) {
        if (evt.candidate) {
            console.log(evt.candidate);
            sendCandidate({type: "candidate",
                              sdpMLineIndex: evt.candidate.sdpMLineIndex,
                              sdpMid: evt.candidate.sdpMid,
                              candidate: evt.candidate.candidate});
        }
    };
    console.log('添加本地视频流...');
    peer.addStream(localStream);
    peer.addEventListener("addstream", onRemoteStreamAdded, false);
    peer.addEventListener("removestream", onRemoteStreamRemoved, false);
    //当接收到远程视频流时，使用本地 video 元素进行显示
    function onRemoteStreamAdded(event) {
        console.log("Added remote stream");
        remoteVideo.src = window.URL.createObjectURL(event.stream);
    }
    //当远程结束通信时，取消本地 video 元素中的显示
    function onRemoteStreamRemoved(event) {
        console.log("移除远程视频流");
        remoteVideo.src = "";
    }
    return peer;
}
```





第 4 步，在 prepareNewConnection()函数中创建 RTCPeerConnection 对象，代码如下所示。目前在浏览器中需要使用供应商前缀。

```
peer = new webkitRTCPeerConnection(pc_config);
```

可以将一个 RTCConfiguration 对象用作 RTCPeerConnection 对象的构造函数的一个参数，该对象用于配置 WebRTC 通信时所使用的 ICE 服务器，代码如下所示。

```
var configuration= { iceServers: [{
                        urls: "stun:stun.services.mozilla.com",
                        username: "louis@mozilla.com",
                        credential: "webrtcdemo"
                    }, {
                        urls: ["stun:stun.example.com", "stun:stun-1.example.com"]
                    }]};
var pc= new RTCPeerConnection(configuration);
```

RTCConfiguration 对象有一个 iceServers 属性，属性值为一个由一个或多个 RTCIceServer 对象所组成的数组。每一个 RTCIceServer 对象代表一个 ICE 服务器，具有如下所示的一些属性。

☑ url：字符串类型，用于指定 WebRTC 通信所使用的 STUN 服务器或 TURN 服务器的 URL 地址，当使用 STUN 服务器或 TURN 服务器时必须指定。

☑ usename：字符串类型，用于指定当访问 STUN 服务器或 TURN 服务器时所使用的用户名，当使用 STUN 服务器或 TURN 服务器时该属性为一个可选的属性。

☑ credential：字符串类型，用于指定当访问 STUN 服务器或 TURN 服务器时所使用的用户密码，当使用 STUN 服务器或 TURN 服务器时该属性为一个可选的属性。

例如，如果需要实现不同计算机中的突破 NAT 网关的 P2P 通信时，可以使用 RTCConfiguration 对象定义 P2P 通信所使用的 STURN 服务器，代码如下所示：

```
var pc_config = {"iceServers":[{"url":"stun:stun.1.google.com:19302"}]};
var peer=null;
try{
    peer = new webkitRTCPeerConnection(pc_config)
}catch(e){
    console.log("建立连接失败，错误:" + e.message);
}
```

在 prepareNewConnection()函数中创建 RTCPeerConnection 对象并将其返回后，需创建 SDP。SDP 包括两种：用于通信呼叫的 Offer、用于应答的 Answer。

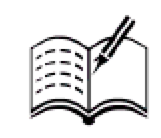
第 5 步，可以使用 RTCPeerConnection 对象的 createOffer()函数创建用于通信呼叫的 Offer。代码可以参考第 2 步代码。createOffer()函数包含 3 个参数：

第一个参数为创建 Offer 成功时调用的回调函数，该函数使用一个参数，参数为创建 Offer 成功时被自动创建的 RTCSessionDescription 对象，该对象代表呼叫方的本地会话描述，具有以下两个属性。

☑ type：一个 RTCSdpType 枚举值，可能的值如下所示。
  - offer：一个呼叫双方的发起方。
  - answer：一个呼叫双方的接收方。
  - pranswer：一个呼叫方做出的临时回答，当接收方给出明确响应后可能被修改。

☑ sdp：一个 SDP 格式的会话描述字符串。





第二个参数为可选参数，参数值为创建 Offer 失败时调用的回调函数。该回调函数有一个参数，参数值为创建 Offer 失败时抛出的错误对象，本例中未使用。

第三个参数为可选参数，参数值为一个 MediaConstraints 对象。该对象具有一个 mandatory 属性，属性值为一个对象，该对象具有一个布尔类型的 OfferToReceiveAudio 属性，属性值为 true 时，表示接收对方传过来的音频信息，属性值为 false 时表示不接收对方传过来的音频信息。该对象同时有一个布尔型的 OfferToReceiveVideo 属性，属性值为 true 时，表示接收对方传过来的视频信息，属性值为 false 时，表示不接收对方传过来的视频信息。

在创建 Offer 成功时所执行的回调函数中，可以使用 RTCPeerConnection 对象的 setLocalDescription()方法保存 WebRTC 连接所使用的本地会话描述，该方法中可以使用三个参数：

☑ 第一个参数为一个代表会话描述的 RTCSessionDescription 对象，该对象是在创建 Offer 时自动设置的本地会话描述。代码如下所示：

```
peerConnection.setLocalDescription(sessionDescription);
```

☑ 第二个参数与第三个参数均为可选参数，参数值分别为保存本地会话描述成功时调用的回调函数和保存本地会话失败时调用的回调函数。

第 6 步，在保存本地会话描述后，可以将本地会话描述发送给对方，在本例中将其显示在“发送的 SDP 字符串:”文本框中，由用户复制给另一个浏览器标签页内的“接收的 SDP 字符串:”文本框中以实现手工发送。代码如下所示：

```
sendSDP(sessionDescription);

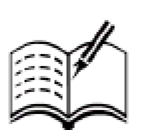
function sendSDP(sdp) {
    var text = JSON.stringify(sdp);
    console.log(text);
    textForSendSDP.value = text;
}
```

第 7 步，在被呼叫方粘贴 SDP 信息，单击“验证 SDP”按钮后，依序调用 onSDP()函数、onOffer()函数和 setOffer()函数，代码如下所示：

```
function onSDP() {
    var text = textToReceiveSDP.value;
    var evt = JSON.parse(text);
    if (peerConnection) {
        onAnswer(evt);
    }else {
        onOffer(evt);
    }
    textToReceiveSDP.value ="";
}
function onOffer(evt) {
    console.log("接收到 offer...")
    console.log(evt);
    setOffer(evt);
    sendAnswer(evt);
}
function setOffer(evt) {
    if (peerConnection) {
```



```
            console.error('peerConnection 已存在!');
        }
        peerConnection = prepareNewConnection();
        peerConnection.setRemoteDescription(new RTCSessionDescription(evt));
    }
```



在 setOffer()函数中，首先调用 prepareNewConnection()函数创建 RTCPeerConnection 对象实例，然后调用该对象的 setRemoteDescription()方法保存接收到的远程会话描述，在本例中为被粘贴在第二个浏览器标签页内“接收的 SDP 字符串:”文本框中的信息。该方法包含三个参数：

☑ 第一个参数为一个会话描述的 RTCSessionDescription 对象，本例利用接收到的远程会话描述来创建该对象。

☑ 第二个参数和第三个参数均为可选参数，参数值分别为保存远程会话描述成功时调用的回调函数，以及保存远程会话失败时调用的回调函数。

第 8 步，在 onOffer()函数中调用 setOffer()函数之后，调用 sendAnswer()函数。sendAnswer()函数代码如下所示：

```
    function sendAnswer(evt) {
        console.log('发送 Answer,创建远程会话描述...' );
        if (! peerConnection) {
            console.error('peerConnection 不存在!');
            return;
        }
        peerConnection.createAnswer(function (sessionDescription) { //成功时调用的回调函数
            peerConnection.setLocalDescription(sessionDescription);
            console.log("发送: SDP");
            console.log(sessionDescription);
            sendSDP(sessionDescription);
        }, function () { //失败时调用的回调函数
            console.log("创建 Answer 失败");
        }, mediaConstraints);
    }
```

在 sendAnswer()函数中，使用 RTCPeerConnection 对象的 createAnswer()方法创建 Answer（被呼叫方的 SDP）。同样，在 createAnswer()方法中可以使用三个参数：

☑ 第一个参数为创建 Answer 成功时调用的回调函数，该函数使用一个参数，参数值为创建 Answer 成功时被自动创建的 RTCSessionDescription 对象，此处该对象代表被呼叫方的本地会话描述。

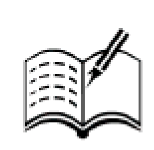
☑ 第二个参数为可选参数，为创建 Answer 失败时调用的回调函数。该回调函数具有一个参数，参数值为创建 Answer 失败时抛出的错误对象。

☑ 第三个参数值为可选参数，参数值为一个 MediaConstraints 对象。

第 9 步，在创建 Answer 成功时调用的回调函数中，同样使用 RTCPeerConnection 对象的 setLocalDescription()方法在被呼叫方保存本地会话描述，然后调用 sendSDP()方法将本地会话描述发送给呼叫方，在本例中将其显示在“发送的 SDP 字符串:”文本框中，由用户复制到另一个浏览器标签页内的“接收的 SDP 字符串:”文本框中。

第 10 步，在呼叫方粘贴 SDP，单击“验证 SDP”按钮后，依序调用 onSDP()函数、onAnswer()函数和 setAnswer()函数。在 setAnswer()函数中调用 RTCPeerConnection 对象的 setRemoteDescription()





方法保存 Answer，代码如下所示：

```
function onSDP() {
    var text = textToReceiveSDP.value;
    var evt = JSON.parse(text);
    if (peerConnection) {
        onAnswer(evt);
    } else {
        onOffer(evt);
    }
    textToReceiveSDP.value ="";
}
function onAnswer(evt) {
    console.log("接收到 Answer...")
    console.log(evt);
    setAnswer(evt);
}
function setAnswer(evt) {
    if (! peerConnection) {
        console.error('peerConnection 不存在!');
        return;
    }
    peerConnection.setRemoteDescription(new RTCSessionDescription(evt));
}
```

这样就完成了呼叫方与被呼叫方的 SDP 信息交换过程。

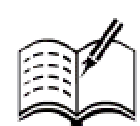
13.2.6 ICE 交换

本节重点分析前面示例的 ICE 交换过程。

第 1 步，首先查看 prepareNewConnection()函数。代码如下所示：

```
function prepareNewConnection() {
    var pc_config = {"iceServers":[]};
    var peer = null;
    try {
        peer = new webkitRTCPeerConnection(pc_config);
    }
    catch (e) {
      console.log("建立连接失败，错误： " + e.message);
    }
    //发送所有 ICE 候选者给对方
    peer.onicecandidate = function (evt) {
        if (evt.candidate) {
            console.log(evt.candidate);
            sendCandidate({type: "candidate",
                              sdpMLineIndex: evt.candidate.sdpMLineIndex,
                              sdpMid: evt.candidate.sdpMid,
                              candidate: evt.candidate.candidate});
        }
    };
```





```
        console.log('添加本地视频流...');
        peer.addStream(localStream);
        peer.addEventListener("addstream", onRemoteStreamAdded, false);
        peer.addEventListener("removestream", onRemoteStreamRemoved, false);
        //当接收到远程视频流时，使用本地 video 元素进行显示
        function onRemoteStreamAdded(event) {
            console.log("Added remote stream");
            remoteVideo.src = window.URL.createObjectURL(event.stream);
        }
        //当远程结束通信时，取消本地 video 元素中的显示
        function onRemoteStreamRemoved(event) {
            console.log("移除远程视频流");
            remoteVideo.src = "";
        }
        return peer;
    }
```

第 2 步，在创建 RTCPeerConnection 对象之后，当 ICE 发现网络候选者，将触发 RTCPeerConnection 对象的 icecandidate 事件。在事件回调函数中，调用 sendCandidate()函数将候选者信息发送给对方，在本例中为将候选者信息显示在“发送的 ICE 字符串:”文本框中，复制到另一个浏览器标签页内“接收的 ICE 字符串:”文本框中以实现手动发送过程，代码如下所示：

```
    function sendCandidate(candidate) {
        var text = JSON.stringify(candidate);
        console.log(text);
        textForSendICE.value = (textForSendICE.value + CR + iceSeparator + CR + text + CR);
        textForSendICE.scrollTop = textForSendICE.scrollHeight;
    }
```

在另一个浏览器标签页内的“接收的 ICE 字符串:”文本框中粘贴 ICE 信息，单击“验证 ICE”按钮，调用 onICE()函数。

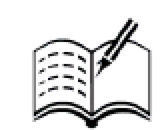
第 3 步，在 onICE()函数中将“接收的 ICE 字符串:”文本框中的信息进行分割，一个个地取出 ICE 候选者信息，再传递给 onCandidate()函数，onICE()函数代码如下所示：

```
    function onICE() {
        var text = textToReceiveICE.value;
        var arr = text.split(iceSeparator);
        for (var i = 1, len = arr.length; i < len; i++) {
            var evt = JSON.parse(arr[i]);
            onCandidate(evt);
        }
        textToReceiveICE.value ="";
    }
```

第 4 步，在 onCandidate()函数中调用 RTCPeerConnection 对象的 addIceCandidate()方法，用于将每一个 ICE 候选者信息传递给 WebRTC 连接对象。代码如下所示：

```
    function onCandidate(evt) {
        var candidate = new RTCIceCandidate({sdpMLineIndex:evt.sdpMLineIndex, sdpMid:evt.sdpMid, candidate:
evt.candidate});
        console.log("接收到 Candidate...")
```

```
        console.log(candidate);
        peerConnection.addIceCandidate(candidate);
    }
```


Note

在呼叫方与被呼叫方均将每一个 ICE 候选者信息都传递给 WebRTC 连接对象之后，双方的 ICE 信息交换完毕。当双方的 SDP 信息及 ICE 信息都交换完毕后，即可开始进行 WebRTC 通信。

第 5 步，在开始通信之后，为了让对方能够接收从本地摄像头中采集的视频数据流，需要使用 WebRTC 连接对象的 addStream()方法，代码如下所示:

```
peer.addStream(localStream);
```

第 6 步，当接收到从对方发送过来的视频流时，触发 WebRTC 连接对象的 addstream 事件。可以在事件回调函数中定义在 video 元素中播放这些视频流，代码如下所示：

```
peer.addEventListener("addstream", onRemoteStreamAdded, false);
//当接收到远程视频流时，使用本地 video 元素进行显示
function onRemoteStreamAdded(event) {
    console.log("Added remote stream");
    remoteVideo.src = window.URL.createObjectURL(event.stream);
}
```

第 7 步，可以使用 WebRTC 连接对象的 close()方法关闭连接，代码如下所示：

```
peerConnection.close();
```

第 8 步，当远程视频流变得不再可用时，将触发 WebRTC 连接对象的 removestream 事件，可以在事件回调函数中定义不再使用本地 video 元素播放这些视频流，代码如下所示：

```
peer.addEventListener("removestream", onRemoteStreamRemoved, false);
//当远程结束通信时，取消本地 video 元素中的显示
function onRemoteStreamRemoved(event) {
    console.log("移除远程视频流");
    remoteVideo.src = "";
}
```

13.3 在线练习

使用 WebRTC 让浏览器实时获取和交换视频、音频和数据。


在线练习



第14章 跨窗口操作

HTML5 新增了多个跨窗口操作的 API，如桌面通知、页面切换可见、全屏显示等。使用这些扩展接口，能够帮助用户摆脱 JavaScript 脚本只能在当前页面发挥作用的限制，增强 Web 应用的适应能力。

【学习重点】

- 正确使用通知 API。
- 基本使用 Page Visibility API。
- 了解 Fullscreen API 的用法。



14.1 通知 API



Note



视频讲解

HTML5 通知 API（Notification API）可以允许在某个事件发生时在桌面向用户显示通知信息，生成的消息不依附于某个标签页，仅仅依附于浏览器。

14.1.1 Notification API 基础

在传统网页设计中，消息推送是基于页面存在的。例如，当用户使用京东进行购物的时候，就无法知道微博有消息推送过来，而只有当用户把当前页面切换到微博网时，才知道有消息推送。

HTML5 通知 API 设计策略：无论用户访问哪个页面，只要有消息，都能推送给用户看到。因此，HTML5 通知生成的消息不依附于某个页面，仅仅依附于浏览器。

提示：Chrome 6+、Opera 23+、Firefox 24+和 Safari 5.2+版本浏览器均支持通知 API。

生成通知的实现步骤如下：

第 1 步，先检查浏览器是否支持 Notification API。

【示例 1】检查浏览器是否支持通知 API，可以通过 window 对象的 Notification 属性判断。

```
if(window.Notification){
    alert("浏览器支持通知 API");
}else{
    alert ("浏览器不支持通知 API");
}
```

线上阅读　视频讲解　示例效果　在线练习

第 2 步，检查浏览器的通知权限，是否允许通知。如果权限不够，则获取浏览器的通知权限。

为了让浏览器可以显示通知，首先要请求让浏览器显示通知的权限。在通知 API 中，使用 Notification 对象的 requestPermission()方法即可，代码如下所示：

```
window.Notification.requestPermission();
```

当 JavaScript 向用户申请让浏览器显示通知的权限时，浏览器显示如图 14.1 所示的提示框。

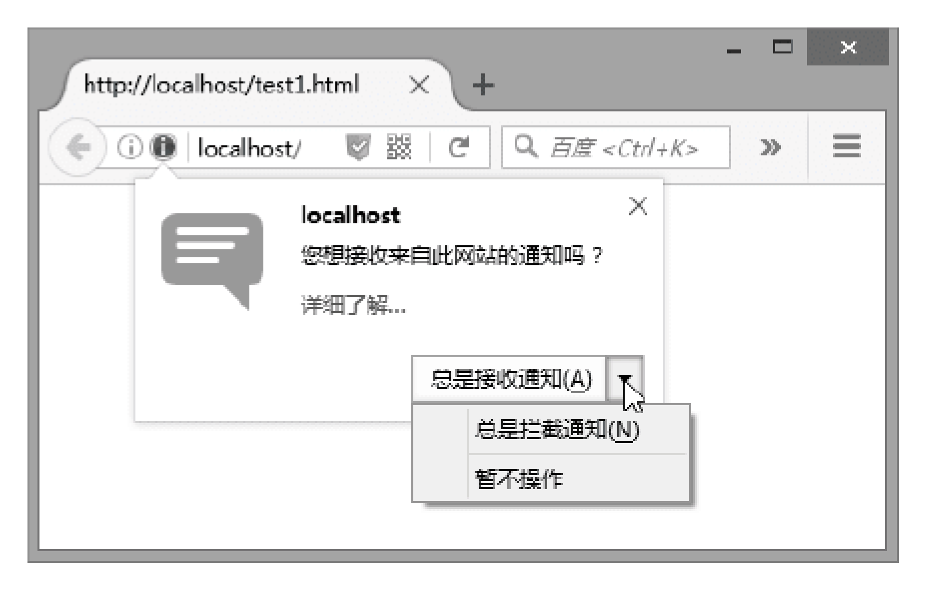


图 14.1　在 Firefox 中申请通知权限

注意：requestPermission()方法只在用户显式触发的事件，如单击按钮、单击鼠标左键或按下键盘上某个键中有效。

【示例 2】用户可以通过 Notification 对象的 permission 属性来判断用户是否给予让浏览器显示通知的权限，代码如下所示。




```
if(window.Notification){
    if (window.Notification.permission == "granted") {
        //获得权限
    }
    else if(window.Notification.permission == "default"){
        window.Notification.requestPermission();  //申请权限
    }
}else{
    alert ("浏览器不支持通知 API");
}
```

Notification 对象的 permission 属性包含 3 个值，说明如下：

☑ default：用户处理结果未知，因此浏览器将视为用户拒绝弹出通知栏。

☑ denied：用户拒绝弹出通知栏。

☑ granted：用户允许弹出通知栏。

第 3 步，创建消息通知。 在获得让浏览器显示通知的权限之后，就可以通过创建 Notification 对象来显示通知，代码如下所示：

```
var notification = new Notification(title,options)
```

该构造函数包含两个参数：第一个参数设置通知的标题；第二个参数为一个对象，用于指定创建通知时可以使用的各种选项，该对象可使用的属性及属性值如下所示：

☑ dir：设置通知中的文字方向，包括 ltr（从左向右）或 rtl（从右向左），默认值为 ltr。

☑ lang：设置通知所使用的语言，属性值必须为一个有效的 BCP 47 语言标识。

☑ body：设置通知中所显示的内容。

☑ tag：设置通知的 ID，即唯一标识符，以区别于其他通知，开发者通过 tag 标识符获取、修改或删除该通知。

☑ icon：设置通知图标，为图片的 URL 地址。

【示例 3】下面代码生成一个通知，定义通知标题、通知图标和通知内容，如图 14.2 所示。

```
<script>
if(window.Notification){
    if (window.Notification.permission == "granted") {
        var notification = new Notification('通知标题', {icon:'images/notice.jpg',body:'通知内容'});
    }
    else if(window.Notification.permission == "default"){
        window.Notification.requestPermission();
    }
}else{
    alert ("浏览器不支持通知 API");
}
</script>
```

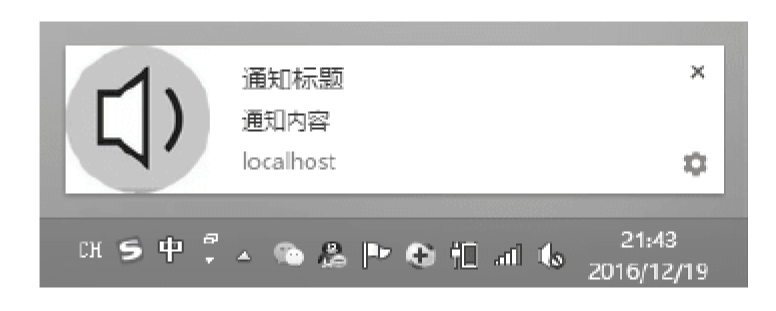


图 14.2　显示通知



Note

第 4 步，监测和管理通知。

Notification 对象提供下面 4 个事件类型，用于监测通知。

☑ show：当通知被显示时触发。
☑ close：当通知被关闭时触发。
☑ click：当通知被单击时触发。
☑ error：当通知引发错误时触发。

另外，使用 Notification 对象的 close()方法可以关闭通知。

【示例 4】下面示例演示了如何使用 Notification 对象事件监测通知。

```
if(window.Notification){
    if (window.Notification.permission == "granted") {
        var notification = new Notification('通知标题', {icon:'images/notice.jpg',body:'通知内容'});
        notification.onshow = function(){
            console.log("显示通知");
        }
        notification.onclose = function(){
            console.log("关闭通知");
        }
        notification.onclick = function(){
            console.log("单击通知");
        }
        notification.onerror = function(Error){
            console.log("通知出错");
        }
    }
    else if(window.Notification.permission == "default"){
        window.Notification.requestPermission();
    }
}else{
    alert ("浏览器不支持通知 API");
}
```



视频讲解

14.1.2　案例：设计桌面通知

下面示例设计当用户单击页面中的控制按钮后，可以开启桌面通知，显示最新微博消息，演示效果如图 14.3 所示。

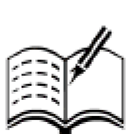


示例效果



图 14.3　手动开启桌面通知





示例主要代码如下所示：

```
<input type="button" value="开启桌面通知" onclick="showNotice();">
<script>
function showNotice(){
    Notification.requestPermission(function(status){
        //status 默认值'default'等同于拒绝，'denied' 意味着用户不想要通知
        //'granted' 意味着用户同意启用通知
        if("granted" != status)
            return;
        var notify = new Notification("澎湃新闻",{
            dir:'auto',
            lang:'zh-CN',
            tag:'sds', //实例化的 notification 的 id
            //icon 支持 ico、png、jpg、jpeg 格式
            icon:'images/pb.jpg',//通知的缩略图
            body:'【保定通报“饭局后驾车撞死人副局长”调查情况：远超醉驾标准】12 月 19 日晚，河
北保定市徐水区委宣传部向澎湃提供最新调查情况通报......' //通知的具体内容
        });
        notify.onclick=function(){    //如果通知消息被单击，通知窗口将被激活
            window.focus();
        }
    });
}
</script>
```

14.1.3　案例：关闭通知



视频讲解

下面示例设计当用户单击页面中的“显示通知”按钮后，可以开启桌面通知，显示最新通知消息，如果单击“关闭通知”按钮，可以关闭通知，演示效果如图 14.4 所示。

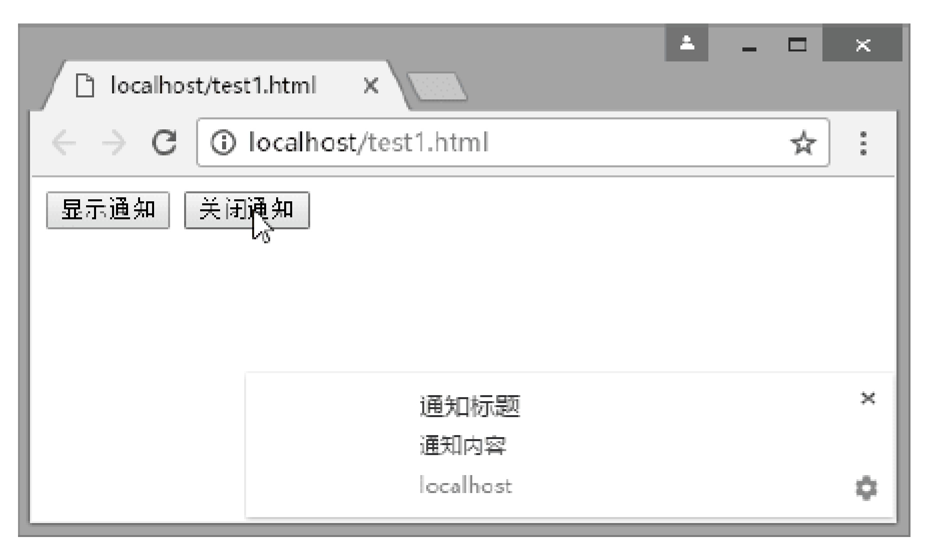


图 14.4　使用脚本关闭通知

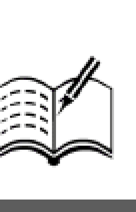


示例效果

示例主要代码如下所示：

```
<script>
var notice;
function createNotification(){
    if (window.Notification.permission == "granted") {
        notice=new Notification('通知标题',
        {icon:' images/pb.jpg ',body:'通知内容'});
        notice.onshow = function() {console.log('通知被显示');};
```







```
            notice.onclose = function() {console.log('通知被关闭');};
        }
        else if(window.Notification.permission == "default"){
            window.Notification.requestPermission();
        }
    }
    function closeNotification(){
        notice.close();
    }
    </script>
    </head>
    <body>
    <button onclick="createNotification()">显示通知</button>
    <button onclick="closeNotification()">关闭通知</button>
```



视频讲解

14.1.4　案例：设计多条通知

下面示例设计当页面显示时，在桌面批量显示 10 条通知，演示效果如图 14.5 所示。



示例效果

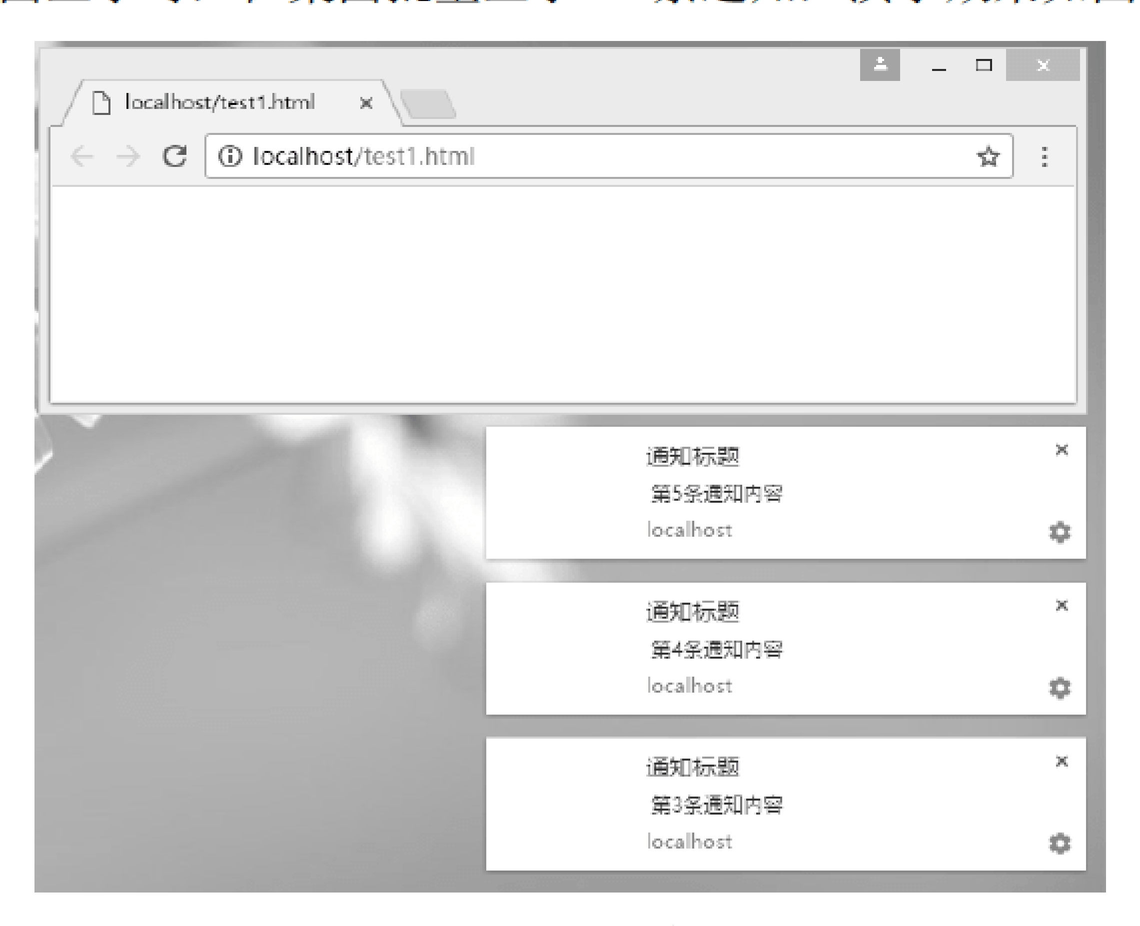


图 14.5　显示多条通知

示例主要代码如下所示：

```
<script>
if (window.Notification.permission == "granted") {
    for(var i=0;i<10;i++)
        var   notice =new Notification('通知标题',{
              icon:'downArrow.gif',
              tag:'MyID' + i ,
              body:'  第'+ i +'条通知内容'
    });
}
else if(window.Notification.permission == "default"){
    window.Notification.requestPermission();
}
</script>
```





14.2　页面可见 API

页面可见 API 可以实现：当与页面进行交互时，如果页面最小化，或者隐藏在其他标签页后面，那么页面中有些功能是可以暂停工作，如轮询服务器或者某些动画效果。

14.2.1　Page Visibility 基础

HTML5 新增 Page Visibility API，它表示页面可见 API。应用 Page Visibility API 之后，在浏览器窗口中只有当前激活的页面处于工作状态，其他隐藏页面将暂停工作，避免不必要的计算，耗费系统资源，干扰用户浏览。

提示：目前 Firefox 1+、Chrome 14+、IE 10+、Opera 12+、Safari 7+版本浏览器支持 Page Visibility API。

【示例 1】在 HTML5 之前，用户可以监听 focus 事件。如果当前窗口获取焦点，那么可以认为用户在与该页面交互，如果失去焦点（blur），那么可以认为用户停止与该页面交互。

```
//当前窗口得到焦点
window.onfocus = function() {
    //开始动画
    //开始 Ajax 轮询等
};
//当前窗口失去焦点
window.onblur = function() {
    //停止动画
    //停止 Ajax 轮询等
};
```

上面设计方法略显简单，如果用户一边打开浏览器看视频，一边在另一个窗口中工作。很显然，焦点集中在工作窗口中，那么浏览器就失去了焦点，而无法正常浏览。Page Visibility API 能够有效帮助用户完全判断，避免不必要的尴尬。

Page Visibility 是一个简单的 API，它包含两个属性和一个事件。

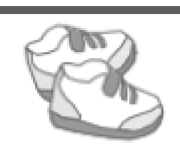
☑ document.hidden：布尔值，表示页面是否隐藏。

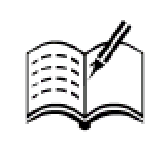
提示：页面隐藏包括：页面在后台标签页中，或者浏览器最小化显示，但是页面被其他软件窗口遮盖并不算隐藏，如打开的 Word 遮住了浏览器。

☑ document.visibilityState：表示当前页面的可见性状态，包括 4 个可能状态值，说明如下。

- hidden：页面在后台标签页中，或者浏览器最小化。
- visible：页面在前台标签页中。
- prerender：页面在屏幕外执行预渲染处理，document.hidden 的值为 true。
- unloaded：页面正在从内存中卸载。

☑ visibilitychange 事件：当文档从可见变为不可见，或者从不可见变为可见时，将触发该事件。





Note

【示例 2】通过监听 visibilitychange 事件，当该事件触发时，获取 document.hidden 的值，根据该值进行页面处理。

```
document.addEventListener('visibilitychange', function(){
    var isHidden = document.hidden;
    if(isHidden) {
        //动画停止
        //服务器轮询停止
    }else {
        //动画开始
        //服务器轮询
    }
});
```

【示例 3】使用 onfocus/onblur 事件可以兼容低版本 IE 浏览器。

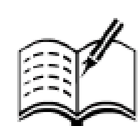
```
(function() {
    var hidden = "hidden";
    //标准用法
    if (hidden in document)
        document.addEventListener("visibilitychange", onchange);
    else if ((hidden = "mozHidden") in document)
        document.addEventListener("mozvisibilitychange", onchange);
    else if ((hidden = "webkitHidden") in document)
        document.addEventListener("webkitvisibilitychange", onchange);
    else if ((hidden = "msHidden") in document)
        document.addEventListener("msvisibilitychange", onchange);
    //兼容 IE9-
    else if ("onfocusin" in document)
        document.onfocusin = document.onfocusout = onchange;
    //兼容其他浏览器
    else
        window.onpageshow = window.onpagehide = window.onfocus = window.onblur = onchange;
    function onchange (evt) {
        var v = "visible", h = "hidden",
                evtMap = {
                    focus:v, focusin:v, pageshow:v, blur:h, focusout:h, pagehide:h
                };
        evt = evt || window.event;
        if (evt.type in evtMap)
            document.body.className = evtMap[evt.type];
        else
            document.body.className = this[hidden] ? "hidden" : "visible";
    }
    //设置初始状态（仅当浏览器支持页面可见性 API）
    if( document[hidden] !== undefined )
        onchange({type: document[hidden] ? "blur" : "focus"});
})();
```





提示：Page Visibility API 适用场景如下：

- ☑ Web 应用拥有幻灯片式的连续播放功能，当页面处于不可见状态时，图片停止播放，当页面变为可见状态时，图片继续播放。
- ☑ 实时显示服务器端信息的应用中，当页面处于不可见状态时，停止定期向服务器端请求数据的处理，当页面变为可见状态时，继续执行定期向服务器端请求数据的处理。
- ☑ 具有播放视频功能的应用中，当页面处于不可见状态时，暂停播放视频，当页面变为可见状态时，继续播放视频。



Note



视频讲解

14.2.2 案例：设计视频页面

本示例使用 Page Visibility 设计当页面被隐藏或最小化显示时，将暂停被播放的视频，同时在标题栏中显示当前暂停播放的时间；当用户切换到当前页面时，再重新播放，标题栏又动态显示播放的进度，演示效果如图 14.6 所示。



（a）动态播放中



（b）暂停播放中



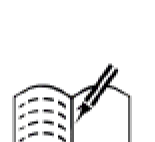
示例效果

图 14.6　在视频页面应用 Page Visibility 技术

示例主要代码如下所示：

```
<video id="videoElement" autoplay controls width="480" height="270">
    <source src="video/chrome.webm" type="video/webm" />
    <source src="video/chrome.ogv" type="video/ogg" />
    <source src="video/chrome.mp4" type="video/mp4; codecs='avc1.42E01E, mp4a.40.2'" />
</video>
<script>
//记录变量，监测视频是否暂停
//视频设置为自动播放
sessionStorage.isPaused = "false";
//设置隐藏属性和可见性变化事件的名称
var hidden, visibilityChange;
if (typeof document.hidden !== "undefined") {
    hidden = "hidden";
    visibilityChange = "visibilitychange";
} else if (typeof document.mozHidden !== "undefined") {
    hidden = "mozHidden";
    visibilityChange = "mozvisibilitychange";
} else if (typeof document.msHidden !== "undefined") {
    hidden = "msHidden";
    visibilityChange = "msvisibilitychange";
} else if (typeof document.webkitHidden !== "undefined") {
    hidden = "webkitHidden";
```





Note

```
        visibilityChange = "webkitvisibilitychange";
    }
    var videoElement = document.getElementById("videoElement");
    //如果该页面是隐藏的，则暂停视频
    //如果显示页面，则播放视频
    function handleVisibilityChange() {
        if (document[hidden]) {
            videoElement.pause();
        } else if (sessionStorage.isPaused !== "true") {
            videoElement.play();
        }
    }
    //如果浏览器不支持 addEventListener 或者页面可见性 API，则进行警告
    if (typeof document.addEventListener === "undefined" ||
        typeof hidden === "undefined") {
        alert("本例需要一个浏览器，如谷歌浏览器，支持页面可见性 API。");
    } else {
        //处理页面可见性变化
        document.addEventListener(visibilityChange, handleVisibilityChange, false);
        //当视频停顿
        videoElement.addEventListener("pause", function(){
            if (!document[hidden]) {
                //如果现在不是因为页面隐藏而暂停，则设置 isPaused 为真
                sessionStorage.isPaused = "true";
            }
        }, false);
        //当视频播放，设置 isPaused 状态
        videoElement.addEventListener("play", function(){
            sessionStorage.isPaused = "false";
        }, false);
        //以当前视频时间设置文档的标题
        videoElement.addEventListener("timeupdate", function(){
            document.title = Math.floor(videoElement.currentTime) + " second(s)";
        }, false);
    }
    </script>
```



视频讲解

14.2.3 案例：设计登录页面

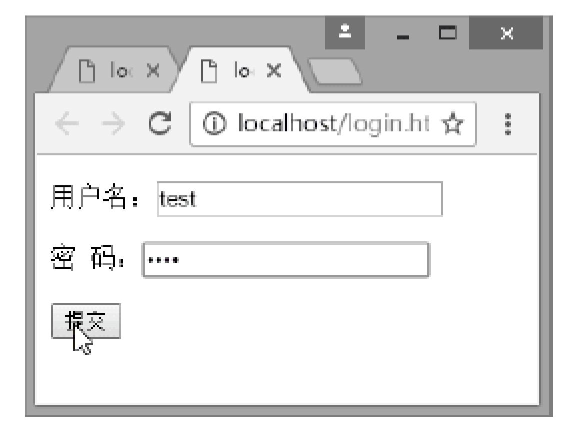
本示例设计当用户在新页面登录成功之后，在其他页面都能够自动呈现登录状态，示例演示效果如图 14.7 所示。



示例效果



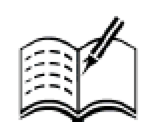
（a）动态显示登录状态



（b）登录页面

图 14.7 在网站登录中应用 Page Visibility 技术





示例主要代码如下所示：

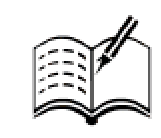
☑　设计首页（test1.html）

```
<script src="pageVisibility.js"></script>
<p id="loginInfo"></p>
<script>
(function() {
    if (typeof pageVisibility.hidden !== "undefined") {
        var eleLoginInfo = document.querySelector("#loginInfo");
        var funLoginInfo = function() {
            var username = localStorage.username || sessionStorage.username;
            if (username) {
                eleLoginInfo.innerHTML = '欢迎回来，<strong>' + username + '</strong>';
                sessionStorage.username = username;
            } else {
                eleLoginInfo.innerHTML = '尚未登录，请<a target="_blank" href="login.html">登录</a>';
            }
        }
        pageVisibility.visibilitychange(function() {
            if (!this.hidden) funLoginInfo();
        });
        funLoginInfo();
        //页面关闭清除 localStorage
        window.addEventListener("unload", function() {
            localStorage.removeItem("username");
        })
    } else {
        alert("浏览器不支持 Page Visibility API");
    }
})();
</script>
```

☑　设计登录页面（login.html）

```
<form id="loginForm" action="" method="post">
    <p>用户名：<input type="text" name="username" required /></p>
    <p>密　码：<input type="password" name="password" required /></p>
    <p><input type="submit" /> </p>
</form>
<script>
(function() {
    if (typeof window.screenX === "number") {
        var eleLoginForm = document.querySelector("#loginForm");
        eleLoginForm.addEventListener("submit", function(e) {
            localStorage.username = document.querySelector("input[name='username']").value;
            alert("登录成功！可以返回前面页面。");
            this.reset();
            e.preventDefault();
        }, false);
    } else {
        alert("浏览器不支持 HTML5 表单");
    }
})();
</script>
```





Note

14.3 全屏 API

FullScreen API 可以通过 JavaScript 脚本向用户请求全屏显示，如果交互完成，随时可以退出全屏状态。

14.3.1 Fullscreen API 基础

HTML5 新增一个 Fullscreen API，可以设计全屏显示模式应用。用户可以通过 DOM 对象的根节点（document.documentElement）或某个元素的 requestFullscreen()方法请求 Fullscreen API。

目前，Firefox 10+、Chrome 16+、Safari 5.1+、Opera 12+、IE 11+版本浏览器支持 Fullscreen API。

【示例 1】下面函数 launchFullscreen()可以根据传入的元素，让该元素全屏显示。

```
function launchFullscreen(element){
    if(element.requestFullscreen) {
        element.requestFullscreen();
    } else if(element.mozRequestFullScreen) {
        element.mozRequestFullScreen();
    } else if(element.msRequestFullscreen){
        element.msRequestFullscreen();
    } else if(element.webkitRequestFullscreen) {
        element.webkitRequestFullScreen();
    }
}
```

注意：最新版本的浏览器都支持 Fullscreen API，但是在使用时需要加上前缀，如 mozRequest FullScreen。使用的时候，可以针对整个网页，也可以针对某个网页元素。

```
launchFullscreen(document.documentElement);
launchFullscreen(document.getElementById("videoElement"));
```

【示例 2】使用 exitFullscreen()或 CancelFullScreen()方法可以取消全屏显示。

```
function exitFullscreen() {
    if (document.exitFullscreen) {
        document.exitFullscreen();
    } else if (document.msExitFullscreen) {
        document.msExitFullscreen();
    } else if (document.mozCancelFullScreen) {
        document.mozCancelFullScreen();
    } else if (document.webkitExitFullscreen) {
        document.webkitExitFullscreen();
    }
}
exitFullscreen();
```

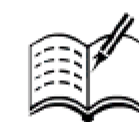
FullScreen API 还定义了两个属性，简单说明如下：

☑ document.fullscreenElement：返回正处于全屏状态的网页元素。



☑　document.fullscreenEnabled：返回一个布尔值，表示当前是否处于全屏状态。

【示例 3】下面代码判断当前页面是否全屏显示，并获取当前全屏显示的元素。



```
var fullscreenEnabled =
    document.fullscreenEnabled ||
    document.mozFullScreenEnabled ||
    document.webkitFullscreenEnabled ||
    document.msFullscreenEnabled;
var fullscreenElement =
    document.fullscreenElement ||
    document.mozFullScreenElement ||
    document.webkitFullscreenElement;
```

在全屏状态下，大多数浏览器的 CSS 支持“:full-screen”伪类，而 IE 11+支持“:fullscreen”伪类。使用这个伪类，可以对全屏状态设置单独的 CSS 样式。

【示例 4】下面样式代码设计全屏模式下的页面样式。

```
<style type="text/css">
:-webkit-full-screen { /*  通用样式  */}
:-moz-full-screen { /*  通用样式  */}
:-ms-fullscreen { /*  通用样式  */}
:full-screen {
    /*  特殊样式  */
    /*  通用样式  */
}
:fullscreen {
    /*  特殊样式  */
    /*  通用样式  */
}
:-webkit-full-screen video {/*  更深层次的元素  */
    width: 100%;
    height: 100%;
}
</style>
```

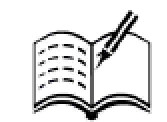
当进入或退出全屏模式时，会触发 fullscreenchange 事件。利用该事件可以监测全屏状态的改变，以便及时做出各种页面响应。

【示例 5】在事件处理函数中，可以通过 DOM 对象的 fullscreen 属性值来判断页面或元素是否处于全屏显示状态。

```
document.addEventListener("fullscreenchange", function () {
    fullscreenState.innerHTML =(document.fullscreen) ? "全屏显示" : "非全屏显示";
    btnFullScreen.value=(document.fullscreen) ? "页面非全屏显示": "页面全屏显示";
}, false);
document.addEventListener("mozfullscreenchange", function () {
    fullscreenState.innerHTML =(document.mozFullScreen) ? "全屏显示" : "非全屏显示";
    btnFullScreen.value=(document.mozFullScreen) ? "页面非全屏显示": "页面全屏显示";
}, false);
document.addEventListener("webkitfullscreenchange", function () {
    fullscreenState.innerHTML =(document.webkitIsFullScreen) ? "全屏显示" : "非全屏显示";
    btnFullScreen.value=(document.webkitIsFullScreen)? "页面非全屏显示": "页面全屏显示";
}, false);
```



在上面代码中，根据不同的浏览器添加浏览器前缀，并将 fullscreen 修改为 FullScreen，例如 mozFullScreen，在 Chrome、Opera 或 Safari 浏览器中需将 fullscreen 改为 webkitlsFullScreen。



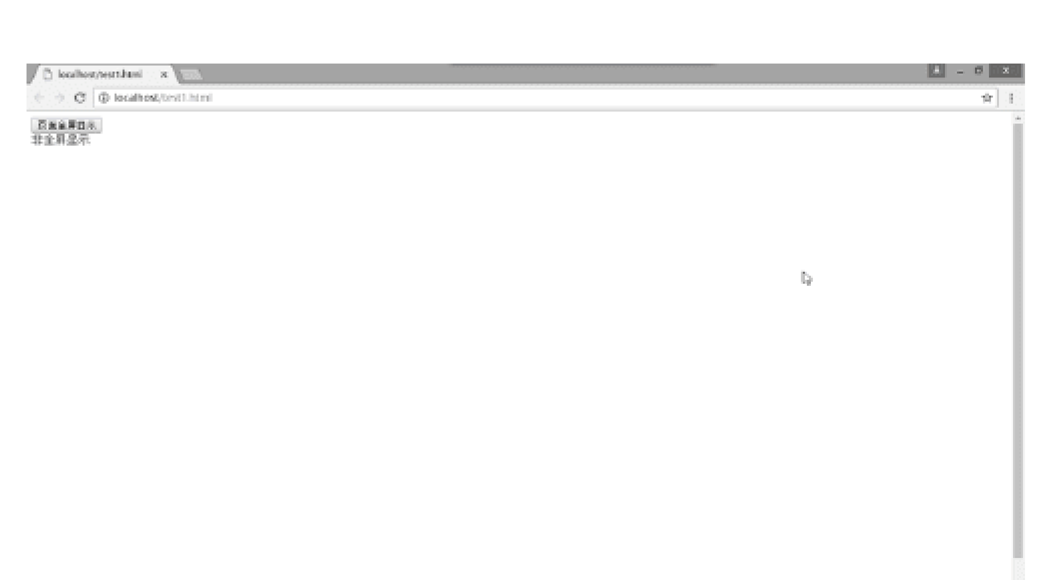
Note



视频讲解

14.3.2 案例：设计全屏显示

本示例设计在页面中显示一个“页面全屏显示”按钮与一个 div 元素，div 元素中显示“非全屏显示”文字。单击“页面全屏显示”按钮后按钮文字变为“页面非全屏显示”，div 元素中显示“全屏显示”文字，页面变为全屏显示状态，页面背景色变为红色。单击“页面非全屏显示”按钮后按钮文字变为“页面全屏显示”，div 元素中显示“非全屏显示”文字，页面恢复为非全屏显示状态，页面背景色恢复为白色。演示效果如图 14.8 所示。



（a）非全屏状态



（b）全屏状态

图 14.8　设计页面全屏显示

示例效果

示例主要代码如下所示：

```
<style type="text/css">
html:-moz-full-screen {background: red;}
html:-webkit-full-screen {background: red;}
html:fullscreen {background: red;}
</style>
<input type="button" id="btnFullScreen" value="页面全屏显示" onclick="toggleFullScreen();">
<div style="width:100%;" id="fullscreenState">非全屏显示</div>
<script type="text/javascript">
var docElm = document.documentElement;
var fullscreenState=document.getElementById("fullscreenState");
var btnFullScreen=document.getElementById("btnFullScreen");
fullscreenState.style.height=docElm.clientHeight+"px";
document.addEventListener("fullscreenchange", function () {
    fullscreenState.innerHTML =(document.fullscreen) ? "全屏显示" : "非全屏显示";
    btnFullScreen.value=(document.fullscreen) ? "页面非全屏显示": "页面全屏显示";
}, false);
document.addEventListener("mozfullscreenchange", function () {
    fullscreenState.innerHTML =(document.mozFullScreen) ? "全屏显示" : "非全屏显示";
    btnFullScreen.value=(document.mozFullScreen) ? "页面非全屏显示": "页面全屏显示";
}, false);
document.addEventListener("webkitfullscreenchange", function () {
    fullscreenState.innerHTML =(document.webkitIsFullScreen) ?"全屏显示" : "非全屏显示";
```



```
        btnFullScreen.value=(document.webkitIsFullScreen) ?"页面非全屏显示": "页面全屏显示";
    }, false);
    function toggleFullScreen(){
        if(btnFullScreen.value=="页面全屏显示"){
            if (docElm.requestFullscreen) {
                docElm.requestFullscreen();
            }
            else if (docElm.mozRequestFullScreen) {
                docElm.mozRequestFullScreen();
            }
            else if (docElm.webkitRequestFullScreen) {
                docElm.webkitRequestFullScreen();
            }
        } else{
            if (document.exitFullscreen) {
                document.exitFullscreen();
            }
            else if (document.mozCancelFullScreen) { //如果为 Firefox 浏览器
                document.mozCancelFullScreen();
            }
            else if (document.webkitCancelFullScreen) {//如果为 Chrome、Opera 或 Safari 浏览器
                document.webkitCancelFullScreen();
            }
        }
    }
    </script>
```



Note



视频讲解

14.3.3　案例：设计全屏播放

本示例设计当按 Enter 键时，视频会自动全屏播放，再次按 Enter 键或者 Esc 键，则退出全屏播放模式，演示效果如图 14.9 所示。



（a）非全屏状态



（b）全屏状态



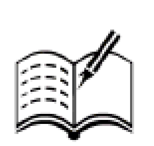
示例效果

图 14.9　设计视频全屏播放

示例主要代码如下所示：

```
    <style type="text/css">
```





Note

```
:-webkit-full-screen #videoElement {/* 使视频拉伸以填充在 WebKit 的屏幕 */
    width: 100%;
    height: 100%;
}
</style>
<p>注意：按回车键切换全屏模式</p>
<video id="videoElement" autoplay controls width="480" height="270">
    <source src="video/chrome.webm" type="video/webm" />
    <source src="video/chrome.ogv" type="video/ogg" />
    <source src="video/chrome.mp4" type="video/mp4; codecs='avc1.42E01E, mp4a.40.2'" />
</video>
<script>
var videoElement = document.getElementById("videoElement");
function toggleFullScreen() {
    if (!document.mozFullScreen && !document.webkitFullScreen) {
        if (videoElement.mozRequestFullScreen) {
            videoElement.mozRequestFullScreen();
        } else {
            videoElement.webkitRequestFullScreen(Element.ALLOW_KEYBOARD_INPUT);
        }
    } else {
        if (document.mozCancelFullScreen) {
            document.mozCancelFullScreen();
        } else {
            document.webkitCancelFullScreen();
        }
    }
}
document.addEventListener("keydown", function(e) {
    if (e.keyCode == 13) {
        toggleFullScreen();
    }
}, false);
</script>
```

14.4 在线练习

练习使用 Web Notifications 新特性。



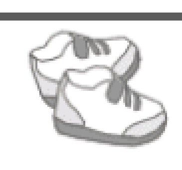
在线练习

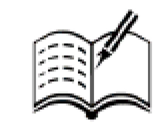
第15章 拖放操作

HTML5 拖放 API 使 Web 应用能够在浏览器中使用拖放功能。例如，用户可以使用鼠标选择可拖动的元素，将元素拖动到可放置的元素内，并通过释放鼠标按钮来放置这些元素。对于 Web 应用程序来说，用户可以自定义能够成为可拖曳的元素类型、可拖曳元素产生的反馈类型，以及可放置的元素。

【学习重点】

- 正确使用拖放 API。
- 应用拖放 API。





Note



视频讲解

15.1　拖放 API 基础

在传统网页设计中，需要借助 JavaScript 的 mousedown、mousemove、mouseup 事件，通过大量脚本来实现拖放操作。HTML5 拖放 API 降低了网页对象拖放的编程难度。本节将详细介绍拖放 API 的基本用法。

15.1.1　拖放功能实现

拖放 API 包含两部分：拖曳（Drag）和释放（Drop），拖曳指的是鼠标点按源对象后一直移动对象不松手，一旦松手即释放了。

读者在学习之前，需要了解两个重要概念：

☑ 源对象：指鼠标点按的一个事物，如一张图片、一个 DIV、一段文本等。

☑ 目标对象：指拖动源对象后移动到一块区域，源对象可以进入这个区域，可以在这个区域上方悬停（未松手），可以释放源对象，将其放置目标对象内（已松手），也可以悬停后离开该区域。

浏览器支持情况：IE 9+、Firefox、Opera 12+、Chrome 和 Safari 5 +。另外，在 Safari 5.1.2 中不支持拖放。

在 HTML5 中，实现拖放操作的步骤如下：

第 1 步，设置源对象的 draggable 属性，设置属性值为 true（draggable="true"），这样就可以启动拖放功能。

提示：img 和 a 元素默认开启了拖放功能，但必须设置 href。

第 2 步，根据 HTML5 拖放 API 定义事件类型，编写与拖放有关的事件处理函数。拖放 API 相关事件说明如表 15.1 所示。

表 15.1　拖放事件

事　件	产生事件的元素	说　明
dragstart	被拖放的元素	开始拖放操作
drag	被拖放的元素	拖放过程中
dragenter	拖放过程中鼠标经过的元素	被拖放的元素开始进入本元素的范围内
dragover	拖放过程中鼠标经过的元素	被拖放的元素正在本元素范围内移动
dragleave	拖放过程中鼠标经过的元素	被拖放的元素离开本元素的范围
drop	拖放的目标元素	有其他元素被拖放到了本元素中
dragend	拖放的对象元素	拖放操作结束

从表 15.1 可以看到，被拖动的源对象可以触发的事件：

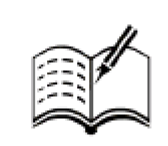
☑ dragstart：源对象开始被拖动。

☑ drag：源对象被拖动过程中，即鼠标可能在移动，也可能未移动。

☑ dragend：源对象被拖动结束。

拖动源对象进入目标对象，在目标对象上可以触发的事件：





☑ dragenter：目标对象被源对象拖动着进入。

☑ dragover：目标对象被源对象拖动着悬停在上方。

☑ dragleave：拖动着源对象离开了目标对象。

☑ drop：拖动着源对象在目标对象上方释放/松手。

【示例】下面示例在页面中插入一个<div id="drag">标签，设置 draggable="true"，启动该元素的拖放功能。同时在页面中插入一个<div id="target">标签，设计为目标对象。本例设计当每次拖放<div id="drag">标签到目标对象<div id="target">标签中时，将在该元素中追加一次提示信息，演示效果如图 15.1 所示。

```
<script type="text/javascript">
function init(){
    var source = document.getElementById("drag");
    var dest = document.getElementById("target");
    source.addEventListener("dragstart", function(ev) {          //（1）拖放开始
        //向 dataTransfer 对象追加数据
        var dt = ev.dataTransfer;
        dt.effectAllowed = 'all';
        //（2）拖动元素为 dt.setData("text/plain", this.id);
        dt.setData("text/plain", "拖入源对象");
    }, false);
    dest.addEventListener("dragend", function(ev) {              //（3）dragend：拖放结束
        ev.preventDefault();                                     //不执行默认处理，拒绝被拖放
    }, false);
    dest.addEventListener("drop", function(ev) {                 //（4）drop：被拖放
        var dt = ev.dataTransfer;                                //从 DataTransfer 对象那里取得数据
        var text = dt.getData("text/plain");
        dest.innerHTML += "<p>" + text + "</p>";
        ev.preventDefault();                                     //（5）不执行默认处理，拒绝被拖放
        ev.stopPropagation();                                    //停止事件传播
    }, false);
}
//（6）设置不执行默认动作，拒绝被拖放
document.ondragover = function(e){e.preventDefault();};
document.ondrop = function(e){e.preventDefault();};
</script>
<style>
#drag { width: 100px; height: 100px; background-color: #93FB40; border-radius: 12px; text-align:center; line-height:100px; color:#F423CC; }
#target { width: 200px; height: 200px; border: 1px dashed gray; margin: -100px 12px 12px; float:right; }
#target  h1 { text-align:center; color:#F423CC; margin:6px 0; font-size:16px; }
</style>

<body onload="init()">
<!--（7）把 draggable 属性设为 true -->
<div id="drag" draggable="true">源对象</div>
<div id="target">
    <h1>目标对象</h1>
</div>
```



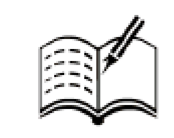


Note

图 15.1　拖放对象

【代码解析】

第 1 步，开始拖动时，触发 dragstart 事件，使用 setData()方法把要拖动的数据存入 DataTransfer 对象。

> 提示：DataTransfer 对象专门用来存放拖放操作时要传递的数据，可以通过拖放事件对象的 dataTransfer 属性进行访问。DataTransfer 对象包含两个重要方法：setData()和 getData()。其中 setData()方法用于向 DataTransfer 对象传递值，而 getData()方法能够从 DataTransfer 对象读取值。

setData()方法的第一个参数为携带数据的数据类型，第二个参数为要携带的数据。第一个参数表示 MIME 类型的字符串，现在支持拖动处理的 MIME 的类型包括以下几种。

☑　"text/plain "：文本文字。
☑　"text/html "：HTML 文字。
☑　"text/xml "：XML 文字。
☑　"text/uri-list "：URL 列表，每个 URL 为一行。

如果把下面代码：

```
dt.setData("text/plain", "拖入源对象");
```

改为：

```
dt.setData("text/plain", this.id);
```

把被拖动元素的 id 作为参数，浏览器在使用 getData()方法读取数据时会自动读取该元素中的数据，所以携带的数据就是被拖动元素中的数据。

第 2 步，针对拖放的目标对象，应该在 dragend 或 dragover 事件内调用事件对象的 preventDefault()方法阻止默认行为。

```
dest.addEventListener("dragend", function(ev) {
    ev.preventDefault();
}, false);
```

第 3 步，目标元素接收到被拖放的元素后，执行 getData()方法从 DataTransfer 对象获取数据。getData()方法包含一个参数，参数为 setData()方法中指定的数据类型，如"text/plain"。

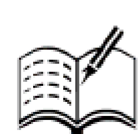
第 4 步，要实现拖放过程，还应在目标元素的 drop 事件中关闭默认处理，否则目标元素不能接收被拖放的元素。



```
dest.addEventListener("drop", function(ev) {
    ev.preventDefault();
    ev.stopPropagation();    //停止事件传播
}, false);
```

第 5 步，要实现拖放过程，还必须设置整个页面为不执行默认处理，否则拖放处理也不能实现。因为页面是先于其他元素接受拖放的，如果页面上拒绝拖放，那么页面上其他元素就都不能接受拖放了。

```
document.ondragover = function(e){e.preventDefault();};
document.ondrop = function(e){e.preventDefault();};
```



Note

15.1.2　DataTransfer 对象



视频讲解

在 15.1.1 节示例中提及 DataTransfer 对象，本节将介绍 DataTransfer 对象的属性和方法，具体说明如表 15.2 所示。

表 15.2　DataTransfer 对象的属性和方法

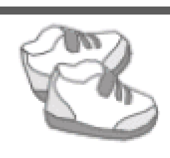
属性/方法	类　型	说　明
dropEffect	属性	表示拖放操作的视觉效果，允许设置值包括：none、copy、link、move。该效果必须在 effectAllowed 属性指定的视觉效果范围内
effectAllowed	属性	指定当元素被拖放时所允许的视觉效果。可以指定的值为：none、copy、copyLink、copyMove、link、linkMove、move、all、uninitialized
types	属性	存入数据的类型，字符串的伪数组
clearData ()	方法	清除 DataTransfer 对象中存放的数据。包含一个参数，设置要清除数据的类型；如果省略参数，则清除全部数据
setData()	方法	向 DataTransfer 对象存入数据，用法参考 15.1.1 节介绍
getData()	方法	从 DataTransfer 对象读取数据，用法参考 15.1.1 节介绍
setDragImage()	方法	设置拖放图标，部分浏览器支持用 canvas 等其他元素来设置，具体说明参考下面介绍

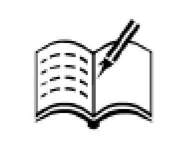
正确使用 DataTransfer 对象的属性和方法，可以实现定制拖放图标，或者定义只支持特定拖放，如复制、移动等，甚至可以实现更复杂的拖放操作。

dropEffect 和 effectAllowed 属性结合起来可以设置拖放时的视觉效果。effectAllowed 属性表示当一个元素被拖动时所允许的视觉效果，一般在 dragstart 事件中定义，可以设置的属性值如表 15.3 所示。

表 15.3　effectAllowed 属性值说明

属　性　值	说　明
copy	允许将被拖动元素复制到拖动的目标元素中
move	允许将被拖动元素移动到拖动的目标元素中





Note

续表

属 性 值	说 明
link	通过拖放操作，被拖动元素会链接到拖动的目标元素上
copyLink	被拖动元素被复制或链接到拖动的目标元素中。根据拖动的目标元素来决定执行复制操作，还是链接操作
copyMove	被拖动元素被复制或移动到拖动的目标元素中。根据拖动的目标元素来决定执行复制操作，还是移动操作
linkMove	被拖动元素被链接或移动到拖动的目标元素中。根据拖动的目标元素来决定执行链接操作，还是移动操作
all	允许执行所有拖动操作，包括复制、移动与链接操作
none	不允许执行任何拖动操作
unintialize	不指定 effectAllowed 属性值。将执行浏览器中默认允许的拖动操作。但是该操作不能通过 effectAllowed 属性值来获取

DataTransfer 对象的 dropEffect 属性表示实际拖放时的视觉效果，一般在 dragover 事件中指定，允许设置的值为 none、copy、link、move。dropEffect 属性所表示的实际视觉效果必须与 effectAllowed 属性值所表示的允许操作相匹配，规则如下所示。

☑ 如果 effectAllowed 属性设置为 none，则不允许拖放元素。
☑ 如果 dropEffect 属性设置为 none，则不允许被拖放到目标元素中。
☑ 如果 effectAllowed 属性设置为 all 或不设置，则 dropEffect 属性允许被设置为任何值。
☑ 如果 effectAllowed 属性设置为具体操作，而 dropEffect 属性也设置了具体视觉效果，则 dropEffect 属性值必须与 effectAllowed 属性值相匹配，否则不允许将被拖放元素拖放到目标元素中。

【示例 1】 下面代码演示了 effectAllowed 和 dropEffect 属性如何配合使用，完整代码可参考上节示例。

```
source.addEventListener("dragstart", function(ev) {
    var dt = ev.dataTransfer;
    dt.effectAllowed = 'copy';
}, false);
dest.addEventListener("dragover", function(ev) {
    var dt = ev.dataTransfer;
    dt.dropEffect = 'copy';
}, false);
```

DataTransfer 对象的 setDragImage()方法包含三个参数：第一个参数设置拖放图标的图标元素，第二个参数设置拖放图标离鼠标指针的 x 轴方向的位移量，第三个参数设置拖放图标离鼠标指针的 y 轴方向的位移量。

【示例 2】 下面代码演示了调用 setDragImage()方法定义拖放图标，演示效果如图 15.2 所示。

```
<script type="text/javascript">
var dragIcon=document.createElement('img');  //创建图标元素
dragIcon.src='images/11.png';                //设置图标来源
function init(){
    var source = document.getElementById("drag");
```





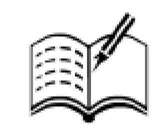
```
            var dest = document.getElementById("target");
            source.addEventListener("dragstart", function(ev) {
                var dt = ev.dataTransfer;
                dt.setDragImage(dragIcon, -10, -10);
                dt.effectAllowed = 'copy';
                dt.setData("text/plain", this.id);
            }, false);
            dest.addEventListener("dragover", function(ev) {
                var dt = ev.dataTransfer;
                dt.dropEffect = 'copy';
            }, false);
            dest.addEventListener("dragend", function(ev) {
                ev.preventDefault();
            }, false);
            dest.addEventListener("drop", function(ev) {
                var dt = ev.dataTransfer;
                var text = dt.getData("text/plain");
                dest.innerHTML += "<p>" + text + "</p>";
                ev.preventDefault();
                ev.stopPropagation();
            }, false);
        }
        document.ondragover = function(e){e.preventDefault();};
        document.ondrop = function(e){e.preventDefault();};
        </script>
        <style>
        #drag { width: 100px; height: 100px; background-color: #93FB40; border-radius: 12px; text-align:center;
line-height:100px; color:#F423CC; }
        #target { width: 200px; height: 200px; border: 1px dashed gray; margin: 12px;}
        </style>

        <body onload="init()">
        <img  id="drag" src="images/1.png" width="314" height="314" alt=""/>
        <div id="target"></div>
```



图 15.2　定义拖放图标效果





Note



视频讲解

15.2 案例实战

下面结合几个案例练习拖放 API 的使用。

15.2.1 设计垃圾箱

本例设计一个简单的垃圾箱，允许用户通过鼠标拖曳的方式把指定的列表项删除，演示效果如图 15.3 所示。





示例效果

图 15.3　设计垃圾箱演示效果

【操作步骤】

第 1 步，新建 HTML5 文档，保存为 test1.html。

第 2 步，构建 HTML 结构，设计一个简单的列表容器，同时模拟一个垃圾箱容器（<div class="dustbin">），<div class="dragremind">为拖曳信息提示框。

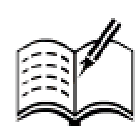
```
<div class="dustbin"><br>
    垃<br>
    圾<br>
    箱</div>
<div class="dragbox">
    <div class="draglist" draggable="true">列表 1</div>
    <div class="draglist" draggable="true">列表 2</div>
    <div class="draglist" draggable="true">列表 3</div>
    <div class="draglist" draggable="true">列表 4</div>
    <div class="draglist" draggable="true">列表 5</div>
    <div class="draglist" draggable="true">列表 6</div>
</div>
<div class="dragremind"></div>
```

第 3 步，在文档头部插入<style>标签，定义内部样式表，设计列表样式和垃圾箱样式。

```
body { font-size: 84%; }
.dustbin { width: 100px; height: 260px; line-height: 1.4; background-color: gray; font-size: 36px; font-family: "微
```



```
软雅黑", "Yahei Mono"; text-align: center; text-shadow: -1px -1px #bbb; float: left; }
    .dragbox { width: 500px; padding-left: 20px; float: left; }
    .draglist { padding: 10px; margin-bottom: 5px; border: 2px dashed #ccc; background-color: #eee; cursor: move; }
    .draglist:hover { border-color: #cad5eb; background-color: #f0f3f9; }
    .dragremind { padding-top: 2em; clear: both; }
```



第 4 步，在页面底部（<body>标签下面）插入<script>标签，定义一个 JavaScript 代码块。输入下面代码定义一个选择器函数。

```
var $ = function(selector) {
    if (!selector) { return []; }
    var arrEle = [];
    if (document.querySelectorAll) {
        arrEle = document.querySelectorAll(selector);
    } else {
        var oAll = document.getElementsByTagName("div"), lAll = oAll.length;
        if (lAll) {
            var i = 0;
            for (i; i<lAll; i+=1) {
                if (/^\./.test(selector)) {
                    if (oAll[i].className === selector.replace(".", "")) {
                        arrEle.push(oAll[i]);
                    }
                } else if(/^#/.test(selector)) {
                    if (oAll[i].id === selector.replace("#", "")) {
                        arrEle.push(oAll[i]);
                    }
                }
            }
        }
    }
    return arrEle;
};
```

第 5 步，获取页面中所有列表项目，然后使用 for 语句逐个为它们绑定 selectstart、dragstart、dragend 事件处理函数。

```
var eleDustbin = $(".dustbin")[0], eleDrags = $(".draglist"), lDrags = eleDrags.length, eleRemind = 
$(".dragremind")[0], eleDrag = null;
for (var i=0; i<lDrags; i+=1) {
    eleDrags[i].onselectstart = function() {
        return false;
    };
    eleDrags[i].ondragstart = function(ev) {
        ev.dataTransfer.effectAllowed = "move";
        ev.dataTransfer.setData("text", ev.target.innerHTML);
        ev.dataTransfer.setDragImage(ev.target, 0, 0);
        eleDrag = ev.target;
        return true;
    };
```



```
        eleDrags[i].ondragend = function(ev) {
            ev.dataTransfer.clearData("text");
            eleDrag = null;
            return false
        };
    }
```


Note

第 6 步，为垃圾箱容器<div class="dustbin">绑定 dragover、dragenter、drop 事件，设计拖曳到垃圾箱上时，高亮显示垃圾箱提示文字；同时当拖入垃圾箱时，删除列表框中对应列表项目；当释放鼠标左键时，在底部<div class="dragremind">容器总显示删除列表项目的提示信息。

```
    eleDustbin.ondragover = function(ev) {
        ev.preventDefault();
        return true;
    };
    eleDustbin.ondragenter = function(ev) {
        this.style.color = "#ffffff";
        return true;
    };
    eleDustbin.ondrop = function(ev) {
        if (eleDrag) {
            eleRemind.innerHTML += '<strong>"' + eleDrag.innerHTML + '"</strong>被扔进了垃圾箱<br>';
            eleDrag.parentNode.removeChild(eleDrag);
        }
        this.style.color = "#000000";
        return false;
    };
```

15.2.2 设计接纳箱

视 频 讲 解

本例设计一个简单的方形盒子，允许用户通过鼠标拖曳方形盒子，并允许把它拖入不同的容器中，演示效果如图 15.4 所示。




线 上 阅 读

图 15.4 设计接纳箱

具体操作步骤请扫码学习。





15.2.3　拖选对象

本例设计一个可视化拖选操作，允许用户通过鼠标拖曳照片，并允许把它拖入不同目标容器中，演示效果如图 15.5 所示。



图 15.5　拖选操作演示效果



具体操作步骤请扫码学习。



15.2.4　可视化删除

本例设计一个文档元素删除操作，允许用户通过鼠标拖曳页面底部的图片，并允许把它拖入不同垃圾桶中删除，演示效果如图 15.6 所示。



图 15.6　可视化删除对象



具体代码解析请扫码学习。

15.3　在 线 练 习



使用 API 进行拖曳行为及相应处理。



第16章 异步交互

XMLHttpRequest 是一个异步交互 API，提供了在客户端与服务器之间数据通信的功能，并且不会刷新页面。2014 年 11 月 W3C 正式发布 XMLHttpRequest Level 2 标准规范，新增了很多实用功能，极大地推动了异步交互在 JavaScript 中的应用。

权威参考：http://www.w3.org/TR/XMLHttpRequest2/

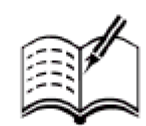


权威参考

【学习重点】

- 使用 responseType 和 response 属性。
- 使用 XMLHttpRequest 发送特殊类型数据。
- 使用 XMLHttpRequest 跨域请求。
- 跟踪文件上传进度。





16.1　XMLHttpRequest 2 基础

老版本的 XMLHttpRequest 插件存在很多缺陷：

☑　只支持文本数据的传送，无法用来读取和上传二进制文件。

☑　传送和接收数据时，没有进度信息，只能提示有没有完成。

☑　受到同域限制，只能向同一域名的服务器请求数据。

XMLHttpRequest 2 做出了大幅改进，简单说明如下：

☑　可以设置 HTTP 请求的时限。

☑　可以使用 FormData 对象管理表单数据。

☑　可以上传文件。

☑　可以请求不同域名下的数据（跨域请求）。

☑　可以获取服务器端的二进制数据。

☑　可以获得数据传输的进度信息。

16.1.1　请求时限

XMLHttpRequest 2 为 XMLHttpRequest 对象新增 timeout 属性，使用该属性可以设置 HTTP 请求时限。

```
xhr.timeout = 3000;
```

上面语句将异步请求的最长等待时间设为 3000 毫秒。超过时限，就自动停止 HTTP 请求。

与之配套的还有一个 timeout 事件，用来指定回调函数。

```
xhr.ontimeout = function(event){
        alert('请求超时！ ');
}
```

16.1.2　FormData 数据对象

XMLHttpRequest 2 新增 FormData 对象，使用它可以处理表单数据。使用方法如下：

第 1 步，新建 FormData 对象。

```
var formData = new FormData();
```

第 2 步，为 FormData 对象添加表单项。

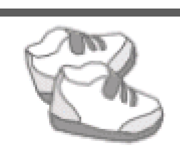
```
formData.append('username', '张三');
formData.append('id', 123456);
```

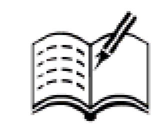
第 3 步，直接传送 FormData 对象。这与提交网页表单的效果完全一样。

```
xhr.send(formData);
```

第 4 步，FormData 对象也可以用来获取网页表单的值。

```
var form = document.getElementById('myform');
```





Note

```
var formData = new FormData(form);
formData.append('secret', '123456'); //添加一个表单项
xhr.open('POST', form.action);
xhr.send(formData);
```

16.1.3 上传文件

新版 XMLHttpRequest 对象不仅可以发送文本信息，还可以上传文件。XMLHttpRequest 的 send() 方法可以发送字符串、Document 对象、表单数据、Blob 对象、文件以及 ArrayBuffer 对象。

【示例】设计一个“选择文件”的表单元素（input[type="file"]），将它装入 FormData 对象。

```
var formData = new FormData();
for (var i = 0; i < files.length;i++) {
    formData.append('files[]', files[i]);
}
```

然后，发送 FormData 对象给服务器。

```
xhr.send(formData);
```

16.1.4 跨域访问

新版本的 XMLHttpRequest 对象，可以向不同域名的服务器发出 HTTP 请求。使用跨域资源共享的前提是：浏览器必须支持这个功能，且服务器端必须同意这种跨域。如果能够满足上面两个条件，则代码的写法与不跨域的请求完全一样。

```
xhr.open('GET', 'http://other.server/and/path/to/script');
```

16.1.5 响应不同类型数据

新版本的 XMLHttpRequest 对象新增 responseType 和 response 属性。

☑ responseType：用于指定服务器端返回数据的数据类型，可用值为 text、arraybuffer、blob、json 或 document。如果将属性值指定为空字符串值或不使用该属性，则该属性值默认为 text。

☑ response：如果向服务器端提交请求成功，则返回响应的数据。

- ➢ responseType 为 text 时，则 response 返回值为一串字符串。
- ➢ responseType 为 arraybuffer 时，则 response 返回值为一个 ArrayBuffer 对象。
- ➢ responseType 为 blob 时，则 response 返回值为一个 Blob 对象。
- ➢ responseType 为 json 时，则 response 返回值为一个 Json 对象。
- ➢ responseType 为 document 时，则 reaponse 返回值为一个 Document 对象。

16.1.6 接收二进制数据

老版本的 XMLHttpRequest 对象只能从服务器接收文本数据，新版本则可以接收二进制数据。

使用新增的 responseType 属性，可以从服务器接收二进制数据。如果服务器返回文本数据，这个属性的值是 text，这是默认值。

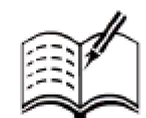
☑ 可以把 responseType 设为 blob，表示服务器传回的是二进制对象。



```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png');
xhr.responseType = 'blob';
```

接收数据的时候，用浏览器自带的 Blob 对象即可。

```
var blob = new Blob([xhr.response], {type: 'image/png'});
```



注意：是读取 xhr.response，而不是 xhr.responseText。

☑ 可以将 responseType 设为 arraybuffer，把二进制数据装在一个数组里。

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png');
xhr.responseType = "arraybuffer";
```

接收数据的时候，需要遍历这个数组。

```
var arrayBuffer = xhr.response;
if (arrayBuffer) {
    var byteArray = new Uint8Array(arrayBuffer);
    for (var i = 0; i < byteArray.byteLength; i++) {
        //执行代码
    }
}
```

16.1.7 监测数据传输进度

新版本的 XMLHttpRequest 对象新增一个 progress 事件，用来返回进度信息。它分成上传和下载两种情况。下载的 progress 事件属于 XMLHttpRequest 对象，上传的 progress 事件属于 XMLHttpRequest.upload 对象。

第 1 步，先定义 progress 事件的回调函数。

```
xhr.onprogress = updateProgress;
xhr.upload.onprogress = updateProgress;
```

第 2 步，在回调函数里面，使用这个事件的一些属性。

```
function updateProgress(event) {
    if (event.lengthComputable) {
        var percentComplete = event.loaded / event.total;
    }
}
```

上面的代码中，event.total 是需要传输的总字节，event.loaded 是已经传输的字节。如果 event.lengthComputable 不为真，则 event.total 等于 0。

与 progress 事件相关的，还有其他五个事件，可以分别指定回调函数：

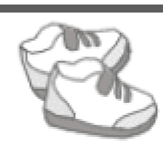
☑ load：传输成功完成。

☑ abort：传输被用户取消。

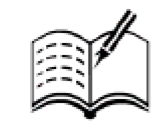
☑ error：传输中出现错误。

☑ loadstart：传输开始。

☑ loadEnd：传输结束，但是不知道成功还是失败。




16.2 案例实战



下面结合具体实例介绍 XMLHttpRequest 2 的应用。

Note

提示：本节示例以 Windows 操作系统+Apache 服务器+PHP 开发语言组合框架为基础进行演示说明。如果读者的本地系统没有搭建 PHP 虚拟服务器，建议先搭建该虚拟环境之后，再详细学习本节内容。



视频讲解

16.2.1 接收 ArrayBuffer 对象

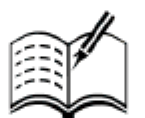
当 XMLHttpRequest 对象的 responseType 属性设置为 arraybuffer 时，服务器端响应数据将是一个 ArrayBuffer 对象。

目前，Firefox 8+、Opera 11.64+、Chrome 10+、Safari 5+和 IE 10+版本浏览器支持将 XMLHttpRequest 对象的 responseType 属性值指定为 arraybuffer。

【示例】下面示例设计在页面中显示一个“下载图片”按钮和一个“显示图片”按钮，单击“下载图片”按钮时，从服务器端下载一幅图片的二进制数据，在得到服务器端响应后创建一个 Blob 对象，并将该图片的二进制数据追加到 Blob 对象中，使用 FileReader 对象的 readAsDataURL()方法将 Blob 对象中保存的原始二进制数据读取为 DataURL 格式的 URL 字符串，然后将其保存在 IndexDB 数据库中。单击“显示图片”时，从 IndexDB 数据库中读取该图片的 DataURL 格式的 URL 字符串，创建一个 img 元素，然后将该 URL 字符串设置为 img 元素的 src 属性值，在页面上显示该图片。

```
<script>
window.indexedDB = window.indexedDB || window.webkitIndexedDB ||
window.mozIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction ||
window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange|| window.webkitIDBKeyRange ||
window.msIDBKeyRange;
window.IDBCursor = window.IDBCursor || window.webkitIDBCursor ||
window.msIDBCursor;
window.URL = window.URL || window.webkitURL;
var dbName = 'imgDB';        //数据库名
var dbVersion = 20170418;    //版本号
var idb;
function init(){
    var dbConnect = indexedDB.open(dbName, dbVersion);    //连接数据库
    dbConnect.onsuccess = function(e){//连接成功
        idb = e.target.result;      //获取数据库
    };
    dbConnect.onerror = function(){alert('数据库连接失败'); };
    dbConnect.onupgradeneeded = function(e){
       idb = e.target.result;
        var tx = e.target.transaction;
        tx.onabort = function(e){
            alert('对象仓库创建失败');
```






```
                };
                var name = 'img';
                var optionalParameters = {
                    keyPath: 'id',
                    autoIncrement: true
                };
                var store = idb.createObjectStore(name,    optionalParameters);
                alert('对象仓库创建成功');
            };
        }
        function downloadPic(){
            var xhr = new XMLHttpRequest();
            xhr.open('GET', 'images/1.png', true);
            xhr.responseType = 'arraybuffer';
            xhr.onload = function(e) {
                if (this.status == 200) {
                    var bb = new Blob([this.response]);
                    var reader = new FileReader();
                    reader.readAsDataURL(bb);
                    reader.onload = function(f) {
                        var result=document.getElementById("result");
                        //在 IndexDB 数据库中保存二进制数据
                        var tx = idb.transaction(['img'],"readwrite");
                        tx.oncomplete = function(){alert('保存数据成功');}
                        tx.onabort = function(){alert('保存数据失败'); }
                        var store = tx.objectStore('img');
                        var value = { img:this.result };
                        store.put(value);
                    }
                }
            };
            xhr.send();
        }
        function showPic(){
            var tx = idb.transaction(['img'],"readonly");
            var store = tx.objectStore('img');
            var req = store.get(1);
            req.onsuccess = function(){
                if(this.result == undefined){
                    alert("没有符合条件的数据");
                } else{
                    var img = document.createElement('img');
                    img.src = this.result.img;
                    document.body.appendChild(img);
                }
            }
            req.onerror = function(){
                alert("获取数据失败");
            }
        }
```



```
</script>
<body onload="init()">
<input type="button" value="下载图片" onclick="downloadPic()"><br/>
<input type="button" value="显示图片" onclick="showPic()"><br/>
<output id="result" ></output>
</body>
```

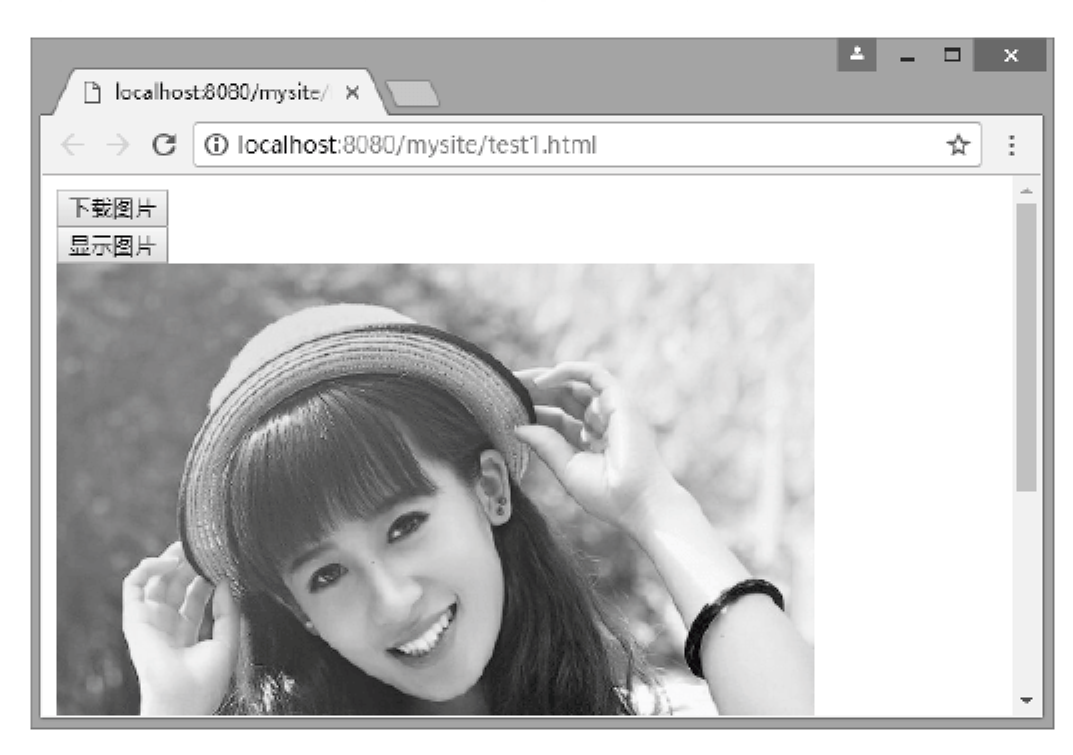
Note

在浏览器中预览，单击页面中“下载图片”按钮，脚本从服务器端下载图片并将该图片二进制数据的 DataURL 格式的 URL 字符串保存在 indexDB 数据库中，保存成功后在弹出提示信息框中显示“保存数据成功”文字，如图 16.1 所示。



图 16.1　下载文件

单击“显示图片”按钮，脚本从 indexDB 数据库中读取图片 DataURL 格式的 URL 字符串，并将其指定为 img 元素的 src 属性值，在页面中显示该图片，如图 16.2 所示。



示 例 效 果

图 16.2　显示照片

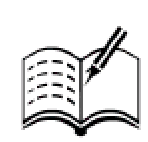
【代码解析】

第 1 步，当用户单击“下载图片”按钮时，调用 downloadPic()函数，在该函数中，XMLHttpRequest 对象从服务器端下载一幅图片的二进制数据，在下载时将该对象的 responseType 属性值指定为 arraybuffer。

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'images/1.png', true);
xhr.responseType = 'arraybuffer';
```

第 2 步，在得到服务器端响应后，使用该图片的二进制数据创建一个 Blob 对象。然后创建一个 FileReader 对象，并且使用 FileReader 对象的 readAsDataURL()方法将 Blob 对象中保存的原始二进制





Note

数据读取为 DataURL 格式的 URL 字符串，然后将其保存在 IndexDB 数据库中。

第 3 步，单击“显示图片”时，从 IndexDB 数据库中读取该图片的 DataURL 格式的 URL 字符串，然后创建一个用于显示图片的 img 元素，将该 URL 字符串设置为 img 元素的 src 属性值，在该页面上显示下载的图片。

16.2.2　接收 Blob 对象



示例效果



视频讲解

当 XMLHttpRequest 对象的 responseType 属性设置为 blob 时，服务器端响应数据将是一个 Blob 对象。目前，Firefox 8+、Chrome 19+、Opera 18+和 IE 10+版本的浏览器支持将 XMLHttpRequest 对象的 responseType 属性值指定为 blob。

【示例】以 16.2.1 节示例为基础，直接修改其中 downloadPic()函数中的代码，设置 xhr.responseType = 'blob'，函数代码如下所示。

```
function downloadPic(){
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'images/1.png', true);
    xhr.responseType = 'blob';
    xhr.onload = function(e) {
        if (this.status == 200) {
            var bb = new Blob([this.response]);
            var reader = new FileReader();
            reader.readAsDataURL(bb);
            reader.onload = function(f) {
                var result=document.getElementById("result");
                //在 IndexDB 数据库中保存二进制数据
                var tx = idb.transaction(['img'],"readwrite");
                tx.oncomplete = function(){alert('保存数据成功');}
                tx.onabort = function(){alert('保存数据失败'); }
                var store = tx.objectStore('img');
                var value = {
                    img:this.result
                };
                store.put(value);
            }
        }
    };
    xhr.send();
}
```

修改完毕后，在浏览器中预览，当在页面中单击“下载图片”按钮和“显示图片”按钮，示例演示效果与上节示例的功能完全一致。

16.2.3　发送字符串

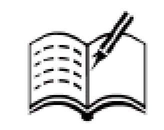


视频讲解

为 XMLHttpRequest 对象设置 responseType = 'text'，可以向服务器发送字符串数据。

【示例】下面示例设计在页面中显示一个文本框和一个按钮，在文本框中输入字符串之后，单击页面上的“发送数据”按钮，将使用 XMLHttpRequest 对象的 send()方法将输入字符串发送到服务器端，在接收到服务器端响应数据后，将该响应数据显示在页面上，演示效果如图 16.3 所示。





示例效果

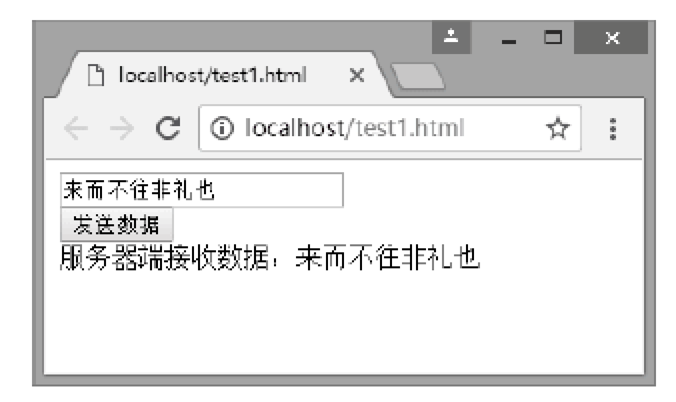


图 16.3　发送字符串演示效果

☑　前台页面（test1.html）

```
<script>
function sendText() {
    var txt=document.getElementById("text1").value;
    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'test.php', true);
    xhr.responseType = 'text';
    xhr.onload = function(e) {
        if (this.status == 200) {
            document.getElementById("result").innerHTML=this.response;
        }
    };
    xhr.send(txt);
}
</script>

<form>
<input type="text" id="text1"><br/>
<input type="button" value="发送数据" onclick="sendText()">
</form>
<output id="result" ></output>
```

☑　后台页面（test.php）

```
<?php
$str =file_get_contents('php://input');
echo '服务器端接收数据：'.$str;
flush();
?>
```



视频讲解

16.2.4　发送表单数据

使用 XMLHttpRequest 对象发送表单数据时，需要创建一个 FormData 对象。用法如下：

```
var form = document.getElementById("forml");
var formData = new FormData(form);
```

FormData()构造函数包含一个参数，表示页面中的一个表单（form）元素。

创建 formData 对象之后，把该对象传递给 XMLHttpRequest 对象的 send()方法即可。用法如下：

```
xhr.send(formData);
```

使用 formData 对象的 append()方法可以追加数据，这些数据将在向服务器端发送数据时随着用户



在表单控件中输入的数据一起发送到服务器端。append()方法的用法如下。

```
formData.append('add_data', '测试');       //在发送之前添加附加数据
```

该方法包含两个参数：第一个参数表示追加数据的键名，第二个参数表示追加数据的键值。

当 formData 对象中包含附加数据时，服务器端将该数据的键名视为一个表单控件的 name 属性值，将该数据的键值视为该表单控件中的数据。

【示例】下面示例在页面中设计一个表单，表单包含一个用于输入姓名的文本框和一个用于输入密码的文本框，以及一个“发送”按钮。输入姓名和密码，单击“发送”按钮，JavaScript 脚本在表单数据中追加附加数据，然后将表单数据发送到服务器端，服务器端接收到表单数据后进行响应，演示效果如图 16.4 所示。

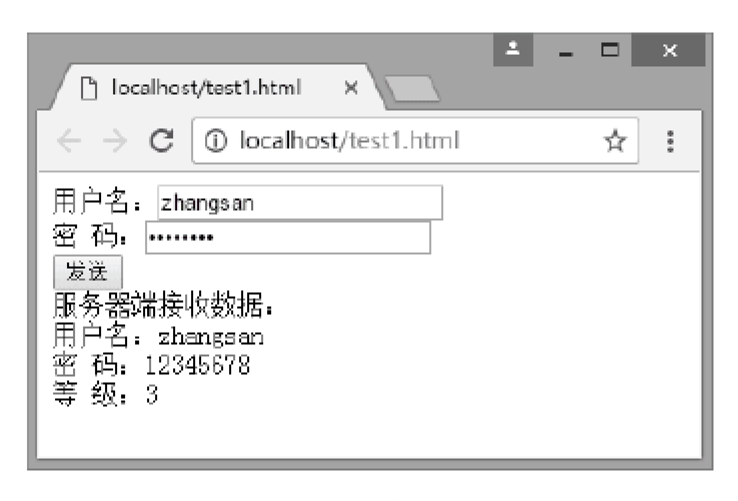


图 16.4　发送表单数据演示效果



示例效果

☑ 前台页面（test1.html）

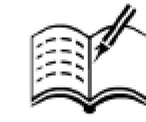
```
<script>
function sendForm() {
    var form=document.getElementById("form1");
    var formData = new FormData(form);
    formData.append('grade', '3'); //在发送之前添加附加数据
    var xhr = new XMLHttpRequest();
    xhr.open('POST','test.php',true);
    xhr.onload = function(e) {
        if (this.status == 200) {
            document.getElementById("result").innerHTML=this.response;
        }
    };
    xhr.send(formData);
}
</script>
<form id="form1">
用户名：<input type="text" name="name"><br/>
密　码：<input type="password" name="pass"><br/>
<input type="button" value="发送" onclick="sendForm();">
</form>
<output id="result" ></output>
```

☑ 后台页面（test.php）

```
<?php
$name =$_POST['name'] ;
$pass =$_POST['pass'] ;
$grade =$_POST['grade'] ;
```



```
echo '服务器端接收数据：<br/>';
echo '用户名：'.$name.'<br/>';
echo '密    码：'.$pass.'<br/>';
echo '等    级：'.$grade;
flush();
?>
```



Note



视频讲解

16.2.5 发送二进制文件

使用 FormData 可以向服务器端发送文件，具体用法：将表单的 enctype 属性值设置为 "multipart/form-data"，然后将需要上传的文件作为附加数据添加到 formData 对象中即可。

【示例】本示例页面中包含一个文件控件和“发送”按钮，使用文件控件在客户端选取一些文件后，单击“发送”按钮，JavaScript 将选取的文件上传到服务器端，服务器端在上传文件成功后将这些文件的文件名作为响应数据返回，客户端接收到响应数据后，将其显示在页面中，演示效果如图 16.5 所示。



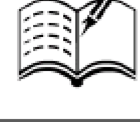
示例效果

图 16.5 发送文件演示效果

☑ 前台页面（test1.html）

```
<script>
function uploadFile() {
    var formData = new FormData();
    var files=document.getElementById("file1").files;
    for (var i = 0;i<files.length;i++) {
        var file=files[i];
        formData.append('myfile[]', file);
    }
    var xhr = new XMLHttpRequest();
    xhr.open('POST','test.php', true);
    xhr.onload = function(e) {
        if (this.status == 200) {
            document.getElementById("result").innerHTML=this.response;
        }
    };
    xhr.send(formData);
}
</script>
<form id="form1" enctype="multipart/form-data">
选择文件<input type="file" id="file1" name="file" multiple><br/>
<input type="button" value="发送" onclick="uploadFile();">
</form>
<output id="result" ></output>
```







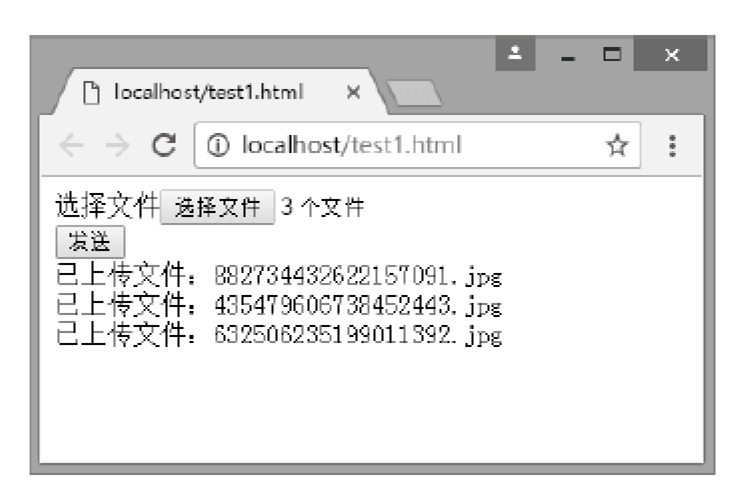
☑　后台页面（test.php）

```
<?php
for ($i=0;$i<count($_FILES['myfile']['name']);$i++) {
    move_uploaded_file($_FILES['myfile']['tmp_name'][$i],'./upload/'.iconv("utf-8","gbk",
$_FILES['myfile']['name'][$i]));
    echo '已上传文件：'.$_FILES['myfile']['name'][$i].'<br/>';
}
flush();
?>
```

16.2.6　发送 Blob 对象

所有 File 对象都是一个 Blob 对象，所以同样可以通过发送 Blob 对象的方法来发送文件。

【示例】下面示例在页面中显示一个“复制文件”按钮和一个进度条（progress 元素），单击“复制文件”按钮后，JavaScript 使用当前页面中所有代码创建一个 Blob 对象，然后通过将该 Blob 对象指定为 XML HttpRequest 对象的 send()方法的参数值的方法向服务器端发送该 Blob 对象，服务器端接收到该 Blob 对象后将其保存为一个文件，文件名为“副本”+当前页面文件的文件名（包括扩展名）。在向服务器端发送 Blob 对象的同时，页面中的进度条将同步显示发送进度，演示效果如图 16.6 所示。





线上阅读

图 16.6　发送 Blob 对象演示效果

具体代码解析请扫码学习。



视频讲解

16.2.7　跨域请求

跨域通信实现方法：在被请求域中提供一个用于响应请求的服务器端脚本文件，并且在服务器端返回响应的响应头信息中添加 Access-Control-Allow-Origin 参数，并且将参数值指定为允许向该页面请求数据的域名+端口号即可。

【示例】下面示例演示了如何实现跨域数据请求。在客户端页面中设计一个操作按钮，当单击该按钮时，向另一个域中的 server.php 脚本文件请求数据，该脚本文件返回一段简单的字符串，本页面接收到该文字后将其显示在页面上，演示效果如图 16.7 所示。



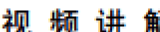


线上阅读

图 16.7　跨域请求数据

具体代码解析请扫码学习。



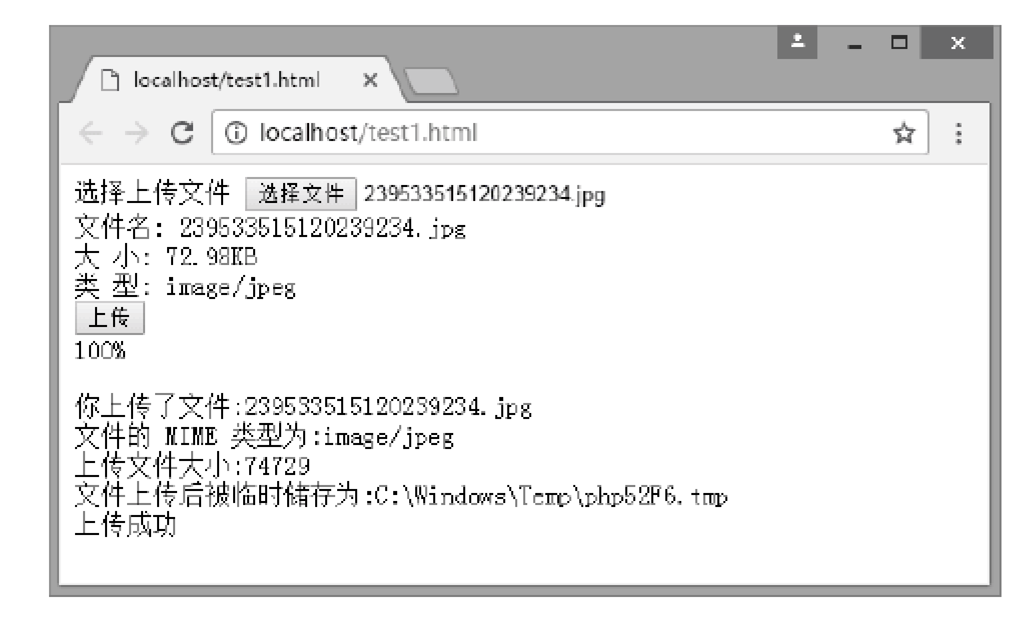

视频讲解



Note

16.2.8 设计文件上传进度条

本例需要 PHP 服务器虚拟环境，同时在站点根目录下新建 upload 文件夹，然后在站点根目录新建前台文件 test1.html，以及后台文件 test.php。在上传文件时，使用 XMLHttpRequest 动态显示文件上传的进度，效果如图 16.8 所示。




线上阅读

图 16.8 上传文件

具体代码解析请扫码学习。

16.3 在线练习

练习 JavaScript 异步通信的应用，培养灵活使用 JavaScript 通信技术实现客户端与服务器无刷新响应的基本能力。


在线练习

第17章

延迟处理

Promise 是一种抽象的异步处理对象，其核心概念为“确保在一件事做完之后，再做另一件事”，这个概念最早出现在 E 语言中，现在 JavaScript 也引入了这个概念，并被纳入 ECMAScript 6 规范。目前，Chrome 34+、Firefox 30+、Opera 20+和 Safari 8+版本浏览器都支持 Promise 对象。

权威参考：https://davidwalsh.name/promises

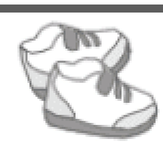
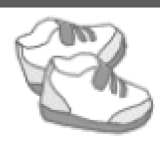


权威参考

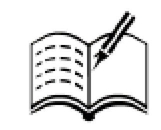
【学习重点】

⏭ 了解回调函数。

⏭ 灵活应用 Promise 对象。

17.1 延迟处理基础



Note



视频讲解

本节将简单介绍回调函数、异步队列和延迟操作对象 Promise。promise 本意是许诺，这里引申为许诺未来特定条件下执行指定回调函数。

17.1.1 从回调函数到异步队列

在 Web 开发中，callback（回调函数）深入人心，它是 JavaScript 执行异步操作的基本方式。

【示例 1】下面示例设计使用 loadImg()自定义函数加载外部图像 a.jpg，如果成功，则再加载 b.jpg，加载 c.jpg。最后，全部加载成功后，在控制台输出提示信息，提示全部加载成功。

```
function loadImg(img,fun){
    var _img = new Image();
    _img.src = img;
    _img.onload = fun;
}
loadImg('images/1.jpg', function() {
    loadImg('images/2.jpg', function() {
        loadImg('images/3.jpg', function() {
            console.log('全部加载完成。');
        });
    });
});
```

上面代码是典型的回调函数金字塔模型，用来设计 JavaScript 异步操作队列。当异步操作的任务很多的时候，维护大量的 callback 将是一场灾难。

【示例 2】下面示例模拟连续执行三个异步操作。在页面中显示一个“读取文件”按钮，当单击按钮时，将读取 1.txt、2.txt、3.txt 三个文本文件，并将读取的文件内容依次显示在页面中。

☑ 1.txt 文件

```
1.立志：昨夜西风凋碧树。独上高楼，望尽天涯路。—— 晏殊的《蝶恋花》
```

☑ 2.txt 文件

```
2.执着：衣带渐宽终不悔，为伊消得人憔悴。—— 柳永的《凤栖梧》
```

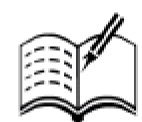
☑ 3.txt 文件

```
3.喜悦：众里寻他千百度，蓦然回首，那人却在灯火阑珊处。—— 辛弃疾的《青玉案·元夕》
```

☑ test2.html

```
<script>
function getData(fileName){
    var xhr = new XMLHttpRequest();
    xhr.open("GET",fileName, true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            if (xhr.status == 200){
```





```
                    var box=document.getElementById("box")
                    box.innerHTML+="<p>" + xhr.responseText + "</p>";
                }
                else    alert("读取文件失败");
            }
        }
        xhr.send();
    }
    function read(){
        getData("1.txt");
        getData("2.txt");
        getData("3.txt");
    }
    </script>
    <input type="button" value="读取文件" onclick="read()"/>
    <div id="box"></div>
```

在浏览器中访问示例页面，单击“读取文件”按钮，脚本将 1.txt、2.txt 和 3.txt 文件中的内容显示在页面中，页面显示效果如图 17.1 所示。

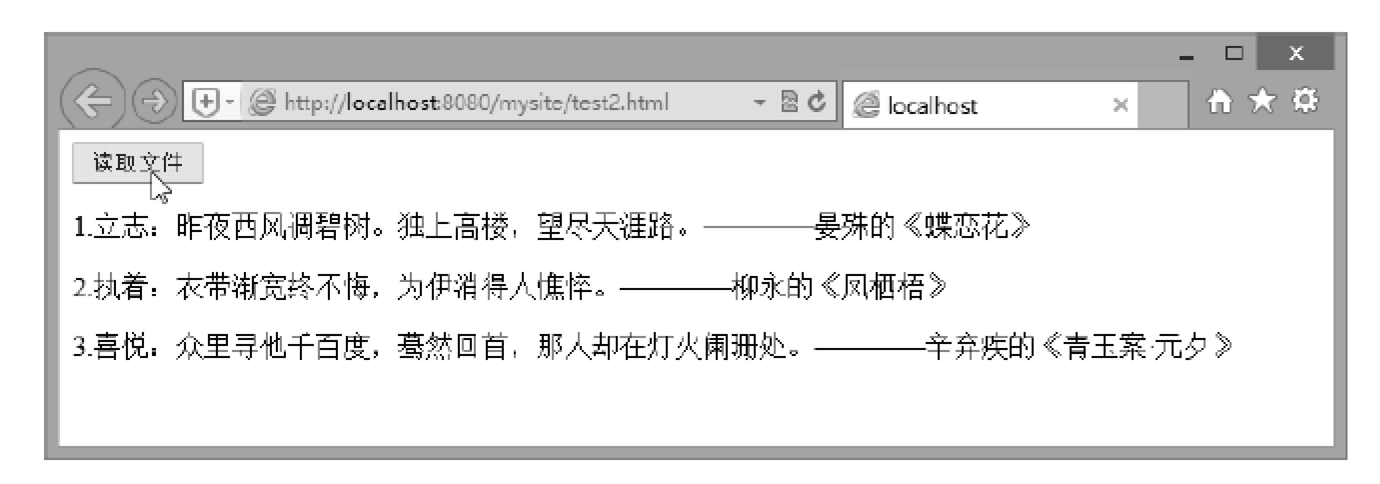


图 17.1　在页面中分别读取文件内容

【示例 3】如果将代码中任一异步读取操作指定为一个不存在的文件，则页面中将显示其他读取成功文件的内容。

```
    function read(){
        getData("1.txt");
        getData("2.txt");
        getData("4.txt");
    }
```

在浏览器中访问页面，单击“读取文件”按钮，则浏览器弹出“读取文件失败”提示信息文字，页面中仍显示脚本读取成功的其他文件内容，如图 17. 2 所示。

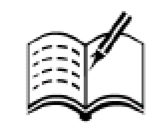


图 17.2　读取最后一个文件失败



现在修改脚本逻辑：设计当读取任意一个文件失败时，任何文件内容均不显示。

【示例 4】下面示例设计全局变量 count 用于记录成功读取文件的个数，初始值为 0，每成功读取到一个文件时将该值加 1，当该值等于 3 时，将读取到的全部文件内容显示在浏览器中，如图 17.3 所示。



```
<script>
function getData(fileName){
    var xhr = new XMLHttpRequest();
    xhr.open("GET",fileName, true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            if (xhr.status == 200){
                count+=1;
                if(count==3){
                    var box=document.getElementById("box")
                    box.innerHTML+="<p>" + xhr.responseText + "</p>";
                }
            }
            else
                alert("读取文件失败");
        }
    }
    xhr.send();
}
function read(){
    var count = 0;
    getData("1.txt");
    getData("2.txt");
    getData("4.txt");
}
</script>
```

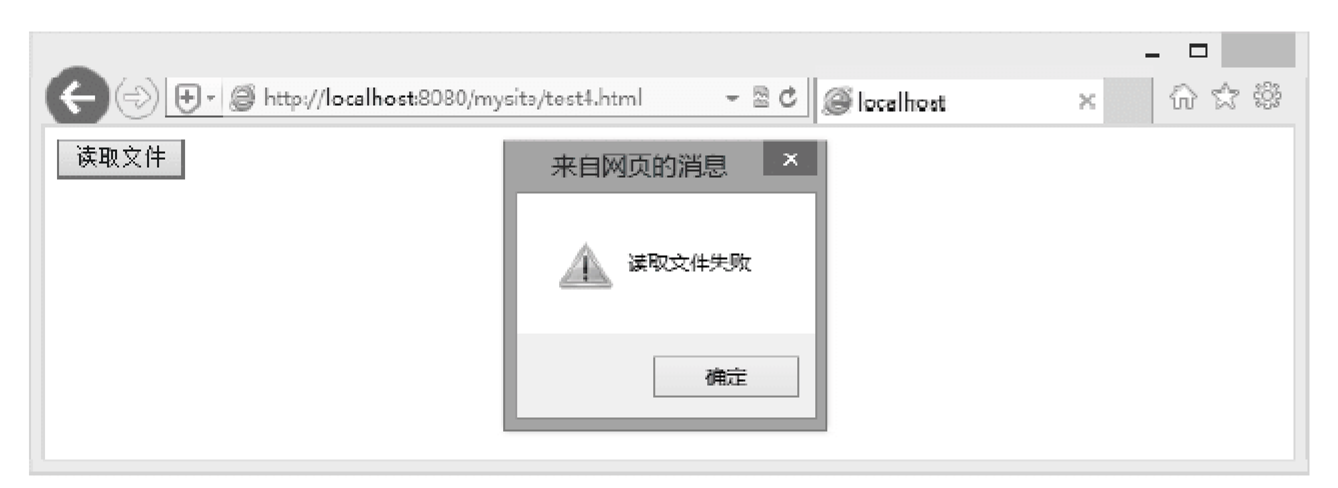
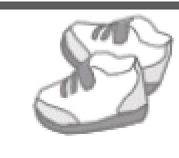


图 17.3　读取文件失败时不再显示全部内容

在应用开发中，经常会遇到“一件事做完之后，再做另一件事”的处理要求，尤其是在设计动画的时候。虽然可以使用回调函数来实现这个要求，但回调函数并不能满足所有情况。

通过本节示例演示，大家能够深切体会到在异步操作中，大量的回调函数队列可能会给开发带来不少麻烦，简单的队列处理，也许能够从容应对，但是在复杂的逻辑环境中，各种关系交织在一起，很容易出错，或者引发逻辑混乱。因此，有必要使用一种全新的编程机制来解决这种重复性的操作，于是 JavaScript Promise API 规范应运而生，切合了人们的痛点需求。





17.1.2　使用 promise 对象

JavaScript Promise API 规范的内容不多，简单描述如下：

☑　一个 promise 对象可能有三种状态：等待（pending）、已完成（fulfilled）、已拒绝（rejected）。

☑　一个 promise 对象的状态只可能从“等待”转到“完成”或者“拒绝”状态，不能逆向转换，同时“完成”和“拒绝”状态不能相互转换。

☑　promise 必须实现 then()方法，then 就是 promise 的核心，而且 then 必须返回一个 promise，同一个 promise 的 then 可以调用多次，并且回调的执行顺序跟它们被定义时的顺序一致。

☑　then()方法接收两个参数，第一个参数是成功时的回调，在 promise 由“等待”转换到“完成”状态时调用；另一个是失败时的回调，在 promise 由“等待”转换到“拒绝”状态时调用。同时，then 可以接受另一个 promise 传入，也接受一个“类 then”的对象或方法，即 thenable 对象。

1．创建许诺

创建 promise 对象的语法格式如下所示：

```
var promise = new Promise(function(resolve, reject){
    //执行异步操作，或者其他代码
    if(/*一切正常*/){
        resolve("一切正常");
    }else{
        reject(Error("处理失败"));
    }
})
```

Promise 类型函数包含一个参数：回调函数。回调函数又包含两个参数：回调函数。如果执行结果正常，则调用 resolve 回调函数，否则调用 reject 回调函数。

提示：在 Promise API 中，将执行结果正常称为 promise 对象返回肯定结果，将执行失败称为 promise 对象返回否定结果。

【示例 1】下面示例新创建一个 promise 对象，定义当图像加载成功时，返回肯定结果；当加载失败时，返回否定结果。

```
var promise = new Promise(function(resolve, reject){
    var _img = new Image();
    _img.src = 'images/1.jpg';
    _img.onload = resolve;
    _img.onerror = reject;
})
```

2．执行许诺

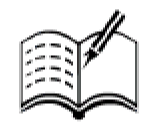
Promise 类型定义了一个原型方法：then()，使用该方法可以执行一个许诺。then()方法的具体用法如下所示：

```
promise.then(resolve, reject)
```

该方法包含两个参数，参数值均为回调函数，简单说明如下：

☑　resolve：设置当返回肯定结果时，调用的回调函数。





Note

☑ reject：设置当返回否定结果时，调用的回调函数。

【示例 2】针对示例 1，在 Promise 类型实例上调用 then()，分别传入 resolve 和 reject 函数，设计当图片加载成功时，弹出“加载完成”提示信息；而当加载失败时，弹出“加载失败”提示信息，如图 17.4 所示。

```
promise.then(resolve,reject) ;
function resolve(){
    alert("加载完成");
}
function reject(){
    alert("加载失败");
}
```



图 17.4　提示加载完成

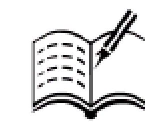
【示例 3】针对 17.1.1 节中示例 2，我们可以使用 promise 对象重新对其进行设计。

```
<script>
function getData(fileName){
    var promise = new Promise(function(ok,err){
        var xhr = new XMLHttpRequest();
        xhr.open("GET",fileName, true);
        xhr.onreadystatechange = function() {
            if (xhr.readyState == 4) {
                if (xhr.status == 200)    ok(xhr.responseText);
                else    err()
            }
        }
        xhr.send();
    })
    promise.then(show, err);
}
function show(info){
    var box=document.getElementById("box")
    box.innerHTML+="<p>" + info + "</p>";
}
function err(){
    alert("读取文件失败");
}
function read(){
    getData("1.txt");
    getData("2.txt");
```



```
        getData("3.txt");
    }
    </script>

    <input type="button" value="读取文件" onclick="read()"/>
    <div id="box"></div>
```



3．连续执行许诺

then()方法返回一个 promise 对象，类似于 jQuery 方法的返回值总是 jQuery 对象一样。因此，用户可以通过链式语法调用 promise 对象的 then()方法，连续运行附加的异步操作。

【示例 4】针对示例 2，我们可以使用下面写法，让提示信息连续提示 3 次。

```
promise.then(resolve,reject).then(resolve,reject).then(resolve,reject);
```

【示例 5】示例 3 没有实际意义，不过用户可以利用 then()让 promise 对象连续执行运算。例如，在实例化 Promise 函数体内，仅调用 resolve()回调函数，然后在调用 then()方法时，仅需要传递一个回调函数参数，在其中执行连续运算，由于实例化的过程，实际上等于创建了一个闭包体，其内变量会长期存在。

```
var promise = new Promise(function(resolve, reject){
    resolve(1);
})
promise.then(function(val){
    alert(val);    //提示 1
    return val+1;
}).then(function(val){
    alert(val);    //提示 2
    return val+1;
}).then(function(val){
    alert (val);   //提示 3
    return val+1;
}).then(function(val){
    alert (val);   //提示 4
});
```

【示例 6】下面示例在页面中设计一个“读取用户资料”按钮及一个表单，单击“读取用户资料”按钮时将从服务器端 user.txt 文件中读取一个用户资料，并将其显示在表单中。

☑　user.txt 文件

```
{
    "user":"张三",
    "password":"123456"
}
```

☑　test6.html

```
<script>
function read(){
    var fileName="user.txt";
    var promise=new Promise(function(resolve, reject) {
        var xhr = new XMLHttpRequest();
```





Note

```
            xhr.open("GET",fileName, true);
            xhr.onreadystatechange = function() {
                if (xhr.readyState == 4) {
                    if (xhr.status == 200) {
                        resolve(xhr.responseText);
                    }else
                        reject();
                }
            }
            xhr.send();
        });
        promise.then(function(response){
            return JSON.parse(response);
        },
        function(){
            alert("读取失败");
        }).then(function(obj){
            document.getElementById("user").value=obj.user;
            document.getElementById("password").value=obj.password;
        });
    }
    </script>
    <form id="form1">
        用户名：<input type="text" id="user" /><br/>
        密　码：<input type="number" id="password" /><br/><br/>
        <input type="button" value="读取资料" onclick="read()"/>
    </form>
```

使用 Chrome 浏览器访问 test6.html 页面，在页面中单击“读取资料”按钮，将从服务器端 user.txt 文件中读取一个用户资料并将其显示在表单中，如图 17.5 所示。

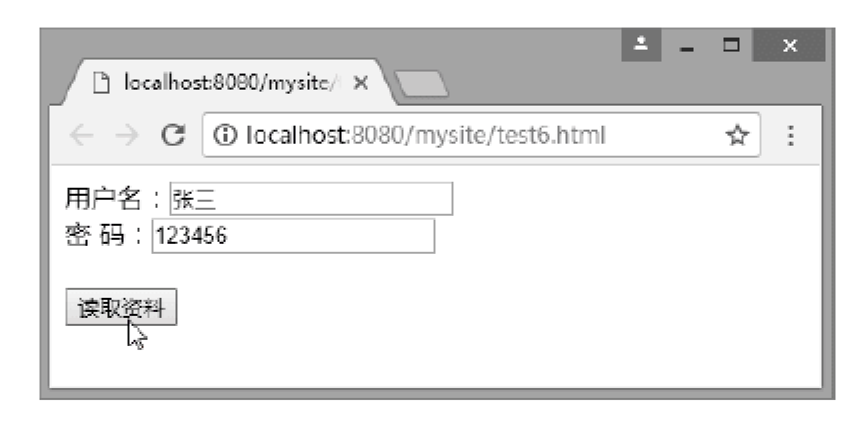


图 17.5　读取用户资料

17.2　案例实战

下面通过示例形式进一步介绍 JavaScript Promise API 复杂用法。

视频讲解

17.2.1　队列操作

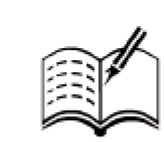
通过 17.1 节示例，我们可以看到 then()方法允许链式语法，使用链式语法可以实现连续操作。

☑　如果调用 then()方法，作为参数的回调函数返回一个值，那么下一个 then()方法将会立即调



用，并且使用该返回值。

☑ 如果调用 then()方法，作为参数的回调函数返回一个 promise 对象，那么下一个 then()方法将对其进行等待，直到这个 promise 对象的回调函数参数的处理结果确定以后才会被调用。



Note

【示例】下面示例设计在页面中显示一个“读取文件”按钮，当单击该按钮时，脚本依次读取 1.txt、2.txt、3.txt 三个文件，并将读取到的文件内容显示在页面中，这三个文本文件的内容可参考第 17.1.1 节示例 2。中途如果读取失败，则所有操作都不会显示，只有全部读取成功之后，才允许全部显示。

```
<script>
function getData(fileName){
    return new Promise(function(resolve, reject) {
        var xhr = new XMLHttpRequest();
        xhr.open("GET",fileName, true);
        xhr.onreadystatechange = function() {
            if (xhr.readyState == 4) {
                if (xhr.status == 200){
                    allData += "<p>" + info + "</p>";
                    resolve();
                }else{
                    alert("读取文件失败");
                }
            }
        }
        xhr.send();
    });
}
function read(){
    allData="";
    getData("1.txt").then(function(){
        return getData("2.txt");
    }).then(function(){
        return getData("3.txt");
    }).then(function(){
        var box=document.getElementById("box");
        box.innerHTML=allData;
    });
}
</script>
<input type="button" value="读取文件" onclick="read()"/>
<div name="box" id="result"></div>
```

在上面脚本中，设计 then()方法的返回值总是 promise 对象，这样强迫 then()链式语法调动，必须总是成功才可以，否则就无法顺利执行后面的 then()方法调用。

17.2.2　异常处理



then()方法包含两个参数：一个为 Promise 处理时，执行成功调用；另一个为 Promise 处理时，执行失败调用。

Note

```
promise.then(function(result){
    console.log(result);      //执行成功
}, function(err){
    console.log(err);         //执行失败
});
```

用户可以使用 catch 捕捉 Promise 处理时抛出的异常，语法格式如下：

```
promise.then(function(result){
    console.log(result);      //执行成功
}).catch(function(err){
    console.log(err);         //执行失败
});
```

上面两段代码具有相同的功能，但是 then()方法的语法更简明，可读性更高。实际上，第二段代码等同于下面这段代码：

```
promise.then(function(result){
    console.log(result);      //执行成功
}).then(undefined,function(err){
    console.log(err);         //执行失败
});
```

promise 对象返回否定结果之后，会跳转到之后第一个配置了否定回调函数的 then()方法，或者是 catch()方法。

对于 then (func1, func2)来说，func1 或 func2 其中之一将会被调用，但绝不会两者都被调用。

对于 then(funcl).catch (func2)来说，当 func1 执行失败时，两者都会被调用，因为它们处于链式语法调用的不同位置。

promise 对象的否定回调函数可以通过 promise.reject()方法显式调用，也可以被 Promise 构造函数的参数值回调函数中抛出的错误隐式调用。

【示例 1】在 Promise 构造函数的参数值回调函数中，将抛出一个异常，该异常将被 catch 机制捕获，从而在浏览器控制台中输出“处理失败!”文字。

```
var jsonPromise = new Promise(function(resolve, reject){
    //当参数值为无效的 JSON 对象时，JSON.parse 将抛出一个错误，导致隐式否定
    resolve(JSON.parse("This isn't JSON"))
})
jsonPromise.then(function(data){
    //下面代码不会被执行
    console.log("处理正常!", data);
}).catch(function(err){
    //下面代码将被执行
    console.log("处理失败", err);
})
```

这种机制将在 Promise 构造器的参数值回调函数中执行所有的 promise 相关工作时，变得非常有用，因为错误将被自动捕捉并转化为否定结果。

【示例 2】在 promise 对象的 then()方法的参数值回调函数中抛出错误。

```
var promise = new Promise(function(resolve, reject){
    resolve();
```



```
})
promise.then(function(){
    var a;
    console.log(a-10);        //抛出未定义错误
}).catch (function(err){
    //下面代码将被执行
    console.log("错误：", err);
});
```



Note

视 频 讲 解

17.2.3　创建序列

在使用 promise 对象序列时，需要用到 Promise 类的静态方法 resolve()。该方法最多可使用一个参数，当参数值为 promise 对象时，resolve()方法根据传入的 promise 对象复制一个新的 promise 对象，如果传入参数为其他任何值，resolve()方法将创建一个以这个值为肯定结果的 promise 对象。如果不指定参数值，创建一个以 undefined 为肯定结果的 promise 对象。

【示例】在下面示例页面中，显示一个文件控件和一个按钮。当使用文件控件选取多个文本文件，并单击按钮时，脚本将读取到的文件内容显示在页面中。

```
<script>
var result=document.getElementById("result");
var file=document.getElementById("file");
var allData="";
function getData(file){
    return new Promise(function(resolve, reject) {
        var reader = new FileReader();
        //将文件以文本形式进行读入页面
        reader.readAsText(file);
        reader.onload = function(f){
            allData+=this.result+"<br/>";
            resolve();
        }
        reader.onerror=function(){
            reject();
        }
    });
}
function get(file){
    return getData(file).catch(function(err){
        alert("读取文件失败");
        throw err;
    });
}
function getSequence(){
    var files=[];
    for(var i=0;i<document.getElementById("file").files.length;i++){
        files.push(document.getElementById("file").files[i]);
    }
    var sequence=Promise.resolve();
    files.forEach(function(file){
```

Note

```
            sequence = sequence.then(function() {
                return get(file);
            });
        });
        return sequence;
    }
    //将文件以文本形式进行读入页面
    function read(){
        Promise.resolve().then(function(){
            return getSequence();
        }).then(function(){
            var result=document.getElementById("result");
            result.innerHTML=allData;
        }).catch(function(){
            console.log("读取文件发生错误");
        });
    }
    </script>
    <div id="divTip"></div>
    <label>选择文件：</label>
    <input type="file" id="file" multiple />
    <input type="button" value="读取文件" onclick="read()"/>
    <div id="result"></div>
```

在浏览器中预览，示例页面中显示一个文件控件和一个控制按钮，当使用文件控件选取多个文本文件，并单击“读取文件”按钮时，脚本将读取到的文件内容显示在页面中，如图 17.6 所示。


示例效果

图 17.6　显示读取到的所有文件内容

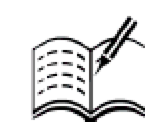
在上面示例中，当单击“读取文件”按钮时，调用 read()函数，在该函数中首先使用 Promise.resolve()方法创建一个 Promise 对象。然后调用该对象的 then()方法，以确保首先调用 getSequence()函数，并且待该函数内的所有异步处理执行完毕后，继续调用该对象的 then()方法在页面中显示用户选取的所有文件内容。

```
    //将文件以文本形式进行读入页面
    function read(){
        Promise.resolve().then(function(){
            return getSequence();
        }).then(function(){
            var result=document.getElementById("result");
            result.innerHTML=allData;
        }).catch(function(){
            console.log("读取文件发生错误");
```



```
        });
    }
```

在 getSequence()函数中，首先根据用户选取的所有文件，创建一个 files 数组。



```
function getSequence(){
    var files=[];
    for(var i=0;i<document.getElementById("file").files.length;i++){
        files.push(document.getElementById("file").files[i]);
    }
```

使用 Promise. resolve()方法创建一个 Promise 对象，并将其赋值给 sequence 变量。

```
var sequence=Promise.resolve();
```

最后，对 files 数组进行遍历，对数组中的每一个文件调用 get()方法以读取文件内容，并将异步处理的执行结果赋值给 sequence 对象，以确保每一个异步处理依序执行。

```
files.forEach(function(file){
    sequence = sequence.then(function() {
        return get(file);
    });
});
return sequence;
```

在 get()方法中，调用 getData()方法以读取文件。如果读取失败，则在浏览器中弹出显示“读取文件失败”提示信息文字并且抛出该错误。

```
function get(file){
    return getData(file).catch(function(err){
        alert("读取文件失败");
        throw err;
    });
}
```

在 getData()方法中，创建一个 Promise 对象以读取每一个文件，如果读取成功，就调用 resolve 参数值回调函数；如果读取失败，就调用 reject 参数值回调函数。

```
function getData(file){
    return new Promise(function(resolve, reject) {
        var reader = new FileReader();
        //将文件以文本形式进行读入页面
        reader.readAsText(file);
        reader.onload = function(f){
            allData+=this.result+"<br/>";
            resolve();
        }
        reader.onerror=function(){
            reject();
        }
    });
}
```



视 频 讲 解



Note

17.2.4 并行处理

使用 Promise 的 all()工具函数可以实现并行执行多个异步处理，all()函数用法如下：

```
Promise.all(arrayOfPromises).then(function(arrayOfResults){
    //回调函数代码
})
```

Promise.all()函数以一个 Promise 对象数组作为参数，并创建一个当所有执行结果都已成功时返回肯定结果的 promise 对象。在该对象的 then()方法中可以得到一个结果数组，无论该对象的肯定结果为何，该结果数组与传入的 promise 对象数组顺序保持一致。

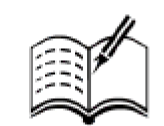
【示例】下面示例演示了 Promise.all()工具函数的应用。当在页面中单击“读取文件”按钮时，调用 read()函数，在该函数中使用 Promise.all()函数读取 1.txt、2.txt、3.txt 文件，传入参数分别为 Promise.all([getData("1.txt"),getData("2.txt"),getData("3.txt")])。

```
<script>
function getData(fileName){
    return new Promise(function(resolve, reject) {
        var xhr = new XMLHttpRequest();
        xhr.open("GET",fileName, true);
        xhr.onreadystatechange = function() {
            if (xhr.readyState == 4) {
                if (xhr.status == 200){
                    resolve(xhr.responseText);
                }else{
                    reject();
                }
            }
        }
        xhr.send();
    });
}
function read(){
    Promise.all([getData("1.txt"),getData("2.txt"),getData("3.txt")])
    .then(function(responses){
        var box=document.getElementById("box");
        responses.forEach(function(response){
            box.innerHTML+="<p>" + response + "</p>";
        });
    },function(){
        alert("读取文件失败");
    });
}
</script>
<input type="button" value="读取文件" onclick="read()"/>
<div id="box"></div>
```

在 getData()函数中将创建并返回一个 promise 对象，该对象的作用为读取文件，读取成功则返回肯定结果，调用 resolve()回调函数，读取失败则返回否定结果，调用 reject()回调函数。



在 read()函数中，待三个 getData()函数中的异步处理全部执行完毕后，调用 Promise.all()函数所创建的 promise 对象的 then()方法，将读取到的文件内容全部显示在页面中，如果任一 getData()函数中创建的 promise 对象返回否定结果，则在浏览器中弹出“读取文件失败”提示信息文字。



提示： JavaScript Promise API 还提供一个 Promise.race()工具函数，该函数同样以一个 promise 对象数组作为参数，但是当数组中的任何元素返回肯定结果时，Promise.race()函数立即返回肯定结果，或者当任何元素返回否定结果时，立即返回否定结果。

17.3　在 线 练 习

练习 Promise 操作。



在 线 练 习



第18章 HTML5 其他 API

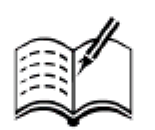
本章将介绍 HTML5 中其他一些各具特色的 API，这些 API 简单，但很实用，浏览器支持情况不是很好，适合读者先了解。

【学习重点】

- 了解鼠标指针锁定 API。
- 掌握 requestAnimationFrame 的基本应用。
- 使用 Mutation Observer。



18.1　指针锁定 API



鼠标指针锁定 API 是一个关于鼠标指针的移动信息（不是鼠标光标的绝对位置信息）的 API。它允许用户获取鼠标指针的移动信息，将鼠标事件锁定到单个目标元素上，消除对于鼠标指针在某个方向上可移动距离的限制，同时从屏幕上移除（本来可见的）鼠标指针。

18.1.1　认识鼠标指针锁定 API

鼠标指针锁定 API 是一个与鼠标信息捕捉相关的 API。鼠标信息捕捉机制将在鼠标指针被拖动的情况下针对一个目标元素的鼠标事件连续提供鼠标指针的相关信息，但是当鼠标左键被松开时，该事件也将被终止。鼠标指针锁定 API 与鼠标信息捕捉机制不同的是：

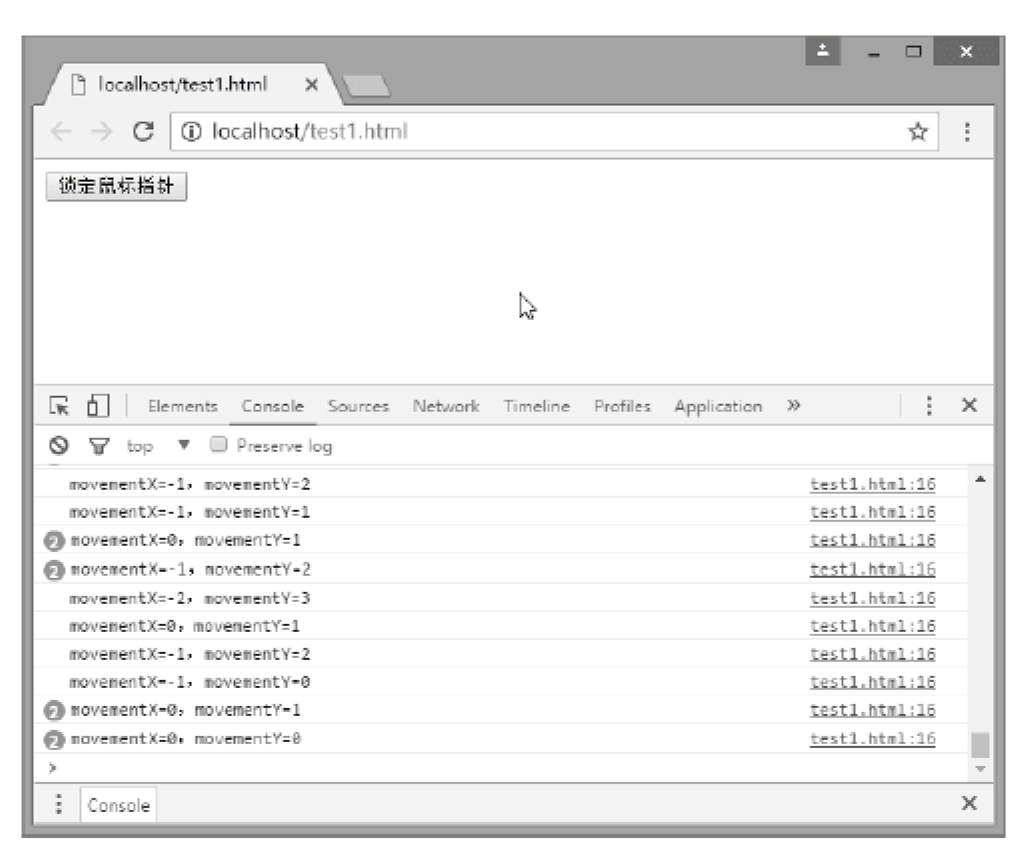
- ☑ 即使鼠标左键已被松开，鼠标指针锁定 API 也不会停止对鼠标指针的获取，除非脚本程序调用该 API 中的方法来显式停止对鼠标指针信息的获取。
- ☑ 鼠标指针锁定 API 不受浏览器或屏幕边界的限制。
- ☑ 当鼠标指针锁定 API 被调用后，不管鼠标左键的状态是什么，它将持续获取鼠标指针信息。
- ☑ 鼠标指针锁定 API 将把光标给隐藏起来。

目前，Firefox 14+、Chrome 22+、Opera 18+版本浏览器对鼠标指针锁定 API 提供支持。

18.1.2　案例：设计全屏鼠标指针锁定

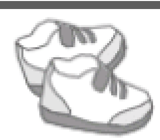
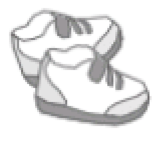
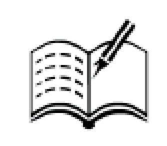


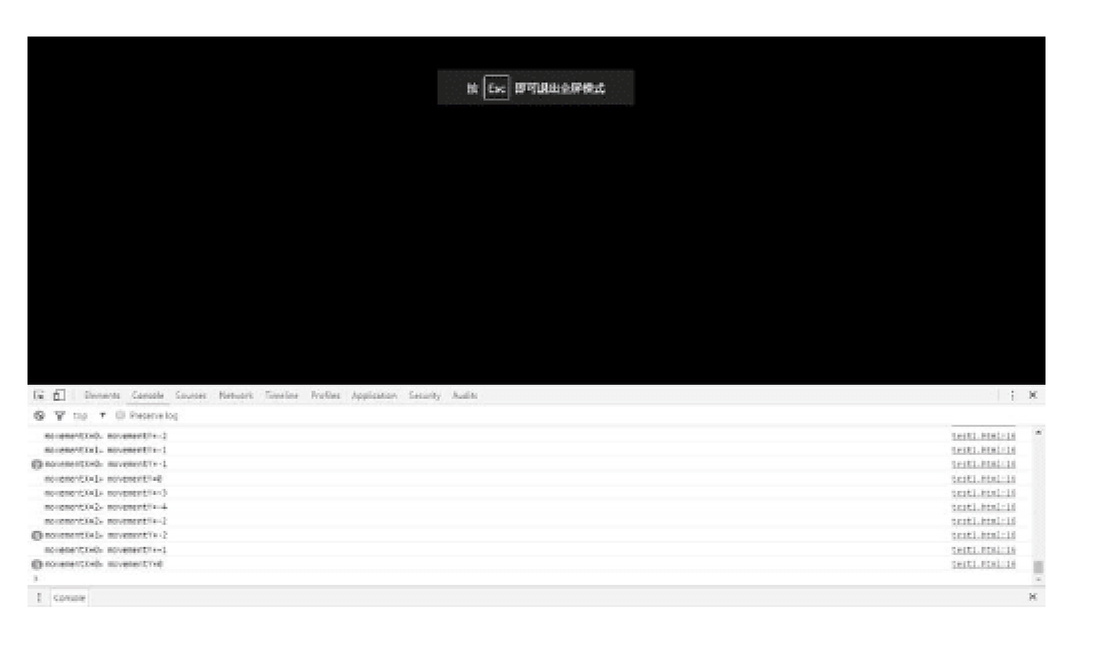
本示例设计在页面中显示一个“锁定鼠标指针”按钮与一个 div 元素。当页面打开时，在控制台中不断输出鼠标指针的移动信息，当用户把鼠标指针移动到浏览器或屏幕之外时，控制台将停止输出信息。如果用户单击“锁定鼠标指针”按钮，JavaScript 脚本将把该 div 元素设定为全屏状态，当用户把鼠标指针移动到浏览器或屏幕之外时，控制台不会停止输出鼠标指针的移动信息。当用户退出全屏状态时，JavaScript 脚本将停止鼠标指针锁定，当用户把鼠标指针移动到浏览器或屏幕之外时，控制台将停止输出信息，演示效果如图 18.1 所示。



（a）非全屏状态

图 18.1　设计全屏鼠标指针锁定



（b）全屏状态

图 18.1　设计全屏鼠标指针锁定（续）

提示：到目前为止，在使用鼠标指针锁定 API 时，需要在锁定目标元素之前，首先将目标元素设定为全屏状态，将来可能会取消这个限制。



线上阅读

具体代码解析请扫码学习。

18.2　requestAnimationFrame

HTML5 新增 window.requestAnimFrame()方法，用于以一种更好的性能来实现帧动画。

18.2.1　认识 requestAnimFrame

在 HTML5+CSS3 时代，设计 Web 动画可以有多种选择，简单说明如下：

☑ 使用 CSS3 的 animattion+keyframes。
☑ 使用 CSS3 的 transition。
☑ 通过 HTML5 的 canvas 作图来实现动画。
☑ 借助 jQuery 动画实现。
☑ 使用 JavaScript 原生的 window.setTimout()或者 window.setInterval()，通过不断更新元素的状态位置等来实现动画，前提是画面的更新频率要达到每秒 60 次才能让肉眼看到流畅的动画效果。

现在 HTML5 新增 window.requestAnimationFrame()方法，该方法用来在页面重绘之前，通知浏览器调用一个指定的函数，以满足开发者操作动画的需求。这个方法接收一个函数为参数，该函数会在重绘前调用。

注意：如果想得到连贯的逐帧动画，在函数中必须重新调用 requestAnimationFrame()。

想做逐帧动画时，应该调用该方法。这就要求用户设计的动画函数执行会先于浏览器重绘动作。通常来说，被调用的频率是每秒 60 次，但是一般会遵循 W3C 标准规定的频率。如果是后台标签页面，重绘频率则会大大降低。用法如下：

```
requestID = window.requestAnimationFrame(callback);    // Firefox 23 / IE10 / Chrome / Safari 7 (incl. iOS)
requestID = window.mozRequestAnimationFrame(callback);     // Firefox < 23
requestID = window.webkitRequestAnimationFrame(callback);  // Older versions Chrome/Webkit
```



参数说明：

callback 在每次需要重新绘制动画时会调用这个参数所指定的函数。这个回调函数会收到一个参数，这个 DOMHighResTimeStamp 类型的参数指示当前时间距离开始触发 requestAnimationFrame 的回调的时间。

返回值 requestID 是一个长整型非零值，作为唯一的标识符，可以将该值作为参数传给 window.cancelAnimationFrame()来取消这个回调函数。



Note

提示：在 Web 动画、APP 动画中，我们经常使用 setInterval 或 setTimeout 定时器修改 DOM、CSS 实现动画，例如：

```
var timer=setInterval(function(){
    //动画
},1000/60)
//清除动画
clearInterval(timer);
```

不过这种方式非常耗费资源，经常会出现动画卡顿现象。

HTML5 的 requestAnimationFrame 方式的优势如下：

☑ 经过浏览器优化，动画更流畅。

☑ 窗口没激活时，动画将停止，节省计算资源。

☑ 更省电，尤其是对移动终端。

requestAnimationFrame 的使用方式如下：

```
function animate() {
    //任意操作
    requestAnimationFrame(animate);
    //做动画
}
//请求动画
requestAnimationFrame(animate);
```

有的时候需要加一些控制，requestAnimationFrame()可以像 setInterval()一样返回一个句柄，然后也可以取消它。控制动画代码如下：

```
var globalID;
function animate() {
    //任意操作
    globalID=requestAnimationFrame(animate);
    //做动画
}
//当加时赛开始
globalID=requestAnimationFrame(animate);
//当停止
cancelAnimationFrame(globalID);
```

目前，Firefox 26+、Chrome 31+、IE 10+、Opera 19+、Safari 6+版本浏览器对 requestAnimationFrame 提供支持。



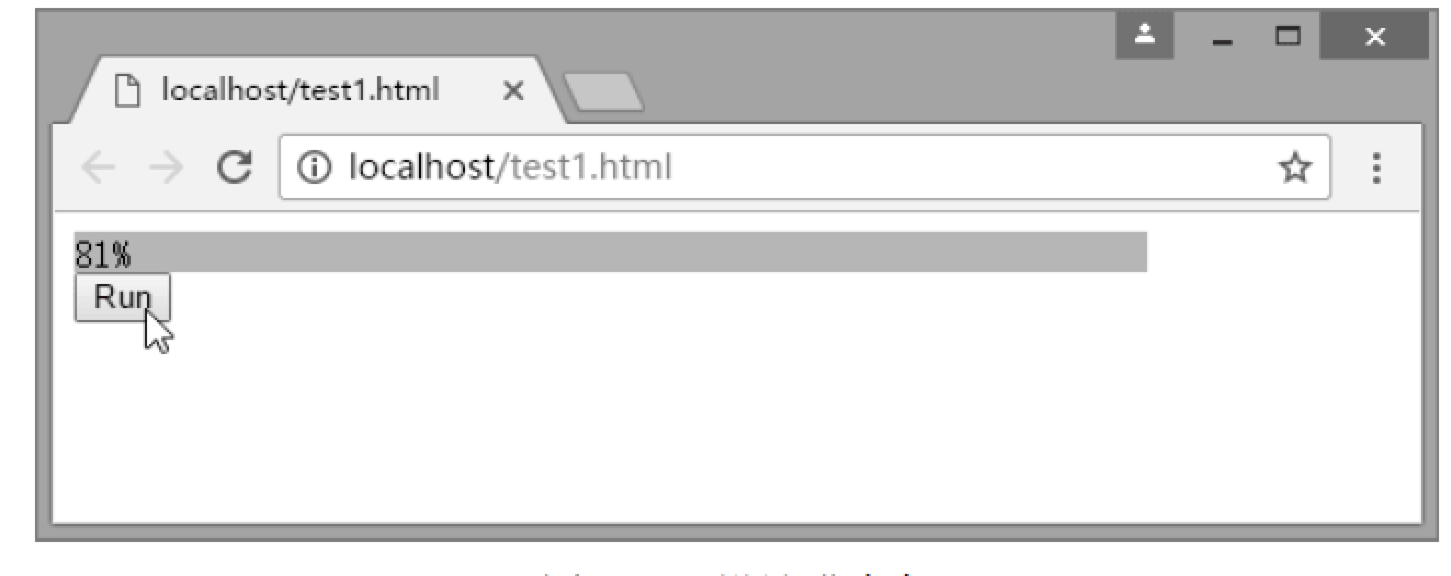


视频讲解

Note

18.2.2　案例：设计进度条

本例模拟一个进度条动画，初始 div 宽度为 1px，在 step()函数中将进度加 1，然后再更新到 div 宽度上，在进度达到 100 之前，一直重复这一过程。为了演示方便加了一个运行按钮，演示效果如图 18.2 所示。



示例效果

图 18.2　设计进度条

示例代码如下所示：

```
<div id="test" style="width:1px;height:17px;background:#0f0;">0%</div>
<input type="button" value="Run" id="run"/>
<script>
window.requestAnimationFrame = window.requestAnimationFrame || window.mozRequestAnimationFrame || window.webkitRequestAnimationFrame || window.msRequestAnimationFrame;
var start = null;
var ele = document.getElementById("test");
var progress = 0;
function step(timestamp) {
    progress += 1;
    ele.style.width = progress + "%";
    ele.innerHTML=progress + "%";
    if (progress < 100) {
        requestAnimationFrame(step);
    }
}
requestAnimationFrame(step);
document.getElementById("run").addEventListener("click", function() {
    ele.style.width = "1px";
    progress = 0;
    requestAnimationFrame(step);
}, false);
</script>
```



视频讲解

18.2.3　案例：设计旋转的小球

本例设计通过 window.requestAnimFrame()方法在 canvas 画布中绘制一个小球运动动画，演示效果如图 18.3 所示。



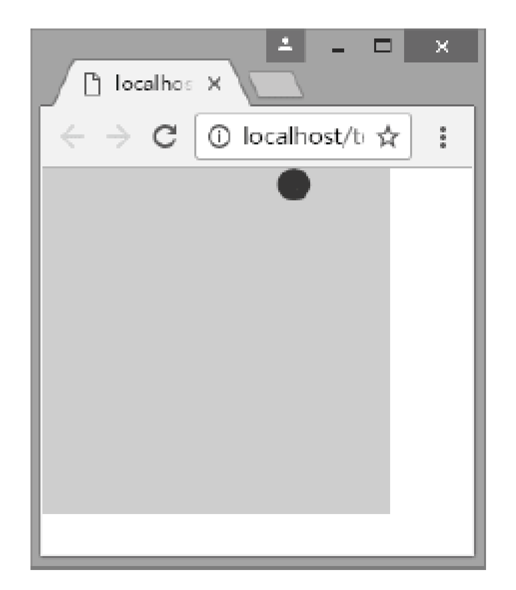


图 18.3　设计旋转的小球动画



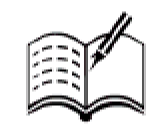
示 例 效 果

Note

示例代码如下所示：

```
<style>body{ margin:0px; padding:0px;}</style>
<script>
window.requestAnimFrame = (function(){
      return   window.requestAnimationFrame             ||
               window.webkitRequestAnimationFrame  ||
               window.mozRequestAnimationFrame        ||
               window.oRequestAnimationFrame           ||
               window.msRequestAnimationFrame          ||
               function(){
                     window.setTimeout(callback, 1000 / 60);
               };
})();
var canvas, context;
init();
animate();
function init() {
      canvas = document.createElement('canvas');
      canvas.style.left=0;
      canvas.style.top=0;
      canvas.width = 210;
      canvas.height = 210;
      context = canvas.getContext('2d');
      document.body.appendChild( canvas );
}
function animate() {
      requestAnimFrame( animate );
      draw();
}
function draw() {
      var time = new Date().getTime() * 0.002;
      var x = Math.sin( time ) * 96 +105;
      var y = Math.cos( time * 0.9 ) * 96 + 105;
      context.fillStyle ='pink';
      context.fillRect( 0, 0, 255, 255 );
      context.fillStyle='rgb(255,0,0)';
      context.beginPath();
      context.arc(x,y,10,0,Math.PI * 2,true);
```

```
        context.closePath();
        context.fill();
    }
</script>
```



Note

18.3 Mutation Observer

Mutation Observer 表示变动观察器，是监视 DOM 变动的接口。当 DOM 对象树发生任何变动时，Mutation Observer 会得到通知。

18.3.1 认识 Mutation Observer

Mutation Observer 类似于事件，可以理解为当 DOM 发生变动会触发 Mutation Observer 事件。但是，它与事件有一个本质不同，比较如下：

☑ 事件是同步触发，也就是说，DOM 发生变动立刻会触发相应的事件。

☑ Mutation Observer 则是异步触发，DOM 发生变动以后，并不会马上触发，而是要等到当前所有 DOM 操作都结束后才触发。

这样设计是为了应付 DOM 变动频繁的情况。例如，如果在文档中连续插入 1000 个段落（p 元素），会连续触发 1000 个插入事件，执行每个事件的回调函数，这很可能造成浏览器的卡顿；而 Mutation Observer 完全不同，只在 1000 个段落都插入结束后才会触发，而且只触发一次。

Mutation Observer 有以下特点：

☑ 等待所有脚本任务完成后才会运行，采用异步方式。

☑ 把 DOM 变动记录封装成一个数组进行处理，而不是一条条地个别处理 DOM 变动。

☑ 可以观察发生在 DOM 节点的所有变动，也可以观察某一类变动。

目前，Chrome 11+、Firefox 16+、IE 11 +、Opera 18+、 Safari 6+版本浏览器对该 API 提供支持。Safari 6.0 和 Chrome 18-25 使用这个 API 的时候，需要加上 WebKit 前缀（WebKitMutationObserver）。可以使用下面的表达式检查浏览器是否支持这个 API。

```
var MutationObserver = window.MutationObserver ||
    window.WebKitMutationObserver ||
    window.MozMutationObserver;
var mutationObserverSupport = !!MutationObserver;
```

使用步骤如下：

第 1 步，使用 MutationObserver()构造函数，新建一个实例，同时指定这个实例的回调函数。

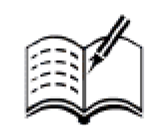
```
var observer = new MutationObserver(callback);
```

第 2 步，使用 observe()方法指定所要观察的 DOM 元素，以及要观察的特定变动。

```
var article = document.querySelector('article');
var options = {
    'childList': true,
    'arrtibutes': true
};
observer.observe(article, options);
```



上面代码首先指定所要观察的 DOM 元素是 article，然后指定所要观察的变动是子元素的变动和属性变动。最后，将这两个限定条件作为参数，传入 observer 对象的 observe()方法。



提示： MutationObserver 所观察的 DOM 变动（即上面代码的 option 对象）包含以下类型，设置值为布尔值：

☑ childList：子元素的变动。
☑ attributes：属性的变动。
☑ characterData：节点内容或节点文本的变动。

想要观察哪一种变动类型，就在 option 对象中指定值为 true。

除了变动类型，option 对象还可以设定以下属性，设置值为布尔值：

☑ attributeOldValue：如果为 true，则表示需要记录变动前的属性值。
☑ characterDataOldValue：如果为 true，则表示需要记录变动前的数据值。
☑ attributesFilter：值为一个数组，表示需要观察的特定属性，如['class', 'str']。
☑ subtree：所有下属节点（包括子节点和子节点的子节点）的变动。

第 3 步，使用 disconnect()方法来停止观察。发生相应变动时，不再调用回调函数。

```
observer.disconnect();
```

第 4 步，使用 takeRecord()方法来清除变动记录，即不再处理未处理的变动。

```
observer.takeRecord
```

提示： DOM 对象每次发生变化，就会生成一条变动记录。这个变动记录对应一个 MutationRecord 对象，该对象包含了与变动相关的所有信息。

MutationRecord 对象包含了 DOM 的相关信息，包含如下属性：

☑ type：观察的变动类型，如 attribute、characterData 或者 childList。
☑ target：发生变动的 DOM 对象。
☑ addedNodes：新增的 DOM 对象。
☑ removeNodes：删除的 DOM 对象。
☑ previousSibling：前一个同级的 DOM 对象，如果没有则返回 null。
☑ nextSibling：下一个同级的 DOM 对象，如果没有就返回 null。
☑ attributeName：发生变动的属性。如果设置了 attributeFilter，则只返回预先指定的属性。
☑ oldValue：变动前的值。这个属性只对 attribute 和 characterData 变动有效，如果发生 childList 变动，则返回 null。

第 5 步，使用 MutationObserve 对象时可能触发的各种事件必须设定的 MutationObserver 选项值说明如下（小括号内的选项为可选选项）。

☑ DOMAttrModified：attributes: true（，attributeOldValue: true）（，attributeFilter:["属性名"]）。
☑ DOMAttributeNameChanged：attributes: true（，attributeOldValue: true）（，attributeFilter:["属性名"]）。
☑ DOMCharactcrDataModified：characterData: true（，characterDataOldValue: true）。
☑ DOMNodeInserted：childList: true（，subtree: true）。



☑ DOMNode insertedIntoDocument：childList: true（，subtree: true）。

☑ DOMNodeRemoved：childList: true（，subtree: true）。

☑ DOMNode RemovedFrom Document：childList: true（，subtree: true）。

☑ DOMSubtreeModified：childList: true，subtree: true。





视 频 讲 解

18.3.2　案例：观察 DOM 元素

本例设计在页面中显示一个 div 元素和一个按钮，单击该按钮时 JavaScript 程序在 div 元素中插入一个 span 元素。另外，在脚本中创建一个 MutationObserver 对象观察 div 元素的变化，将该对象的 observe()方法中的第二个参数值对象的 childList 属性值设置为 true，以观察 div 元素的子元素的变化（包括在 div 元素中插入子元素的操作），当观察到 div 元素中插入子元素时在浏览器中弹出显示“检测到 DOM 变化”提示信息文字，演示效果如图 18.4 所示。



示 例 效 果

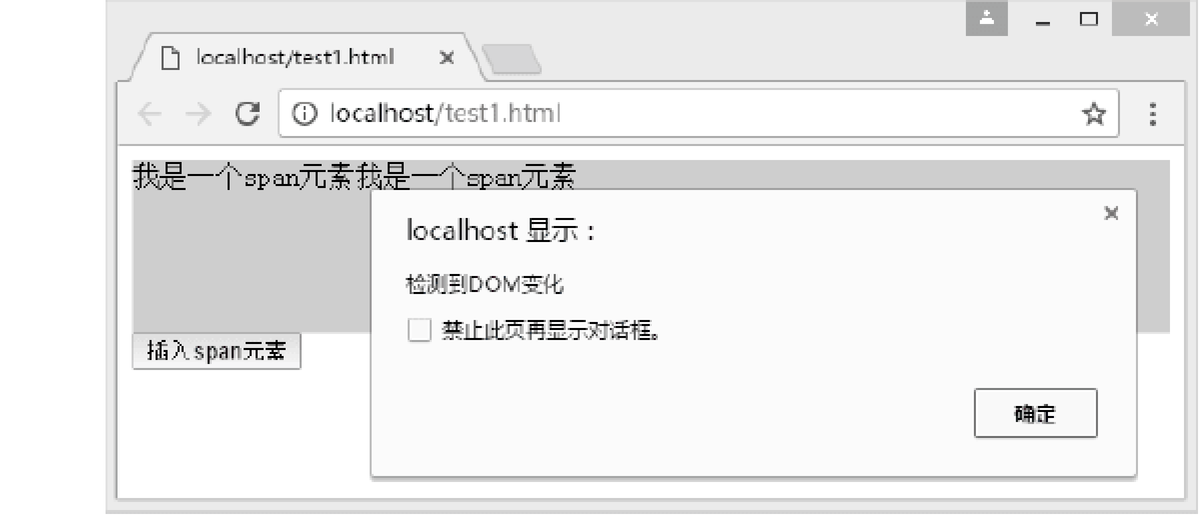


图 18.4　观察 DOM 元素变化

示例代码如下所示：

```
<div id="div" style="height: 100px;width:100%;background-color:pink;"></div>
<input type="button" value="插入 span 元素" onclick="changeDiv();">
<script type="text/javascript">
function onchange(mutationRecords,mutationObserver) {
    alert("检测到 DOM 变化");
    console.log(mutationRecords);
    console.log(mutationObserver);
}
var div = document.getElementById('div');
var mo = new window.MutationObserver(onchange),
options = {childList:true};
mo.observe(div,options);
function changeDiv(){
    var span=document.createElement("span");
    span.innerHTML="我是一个 span 元素";
    div.appendChild(span);
}
</script>
```



视 频 讲 解

18.3.3　案例：观察 DOM 属性

本例设计在页面中显示两个 a 元素和一个按钮，单击该按钮时，会同时修改两个 a 元素的 href



属性值。使用 MutationObserver 对象来观察两个 a 元素，当单击“修改 a 元素”按钮时，在控制台中输出两个 a 元素被修改的属性名及修改前的 href 属性值，演示效果如图 18.5 所示。

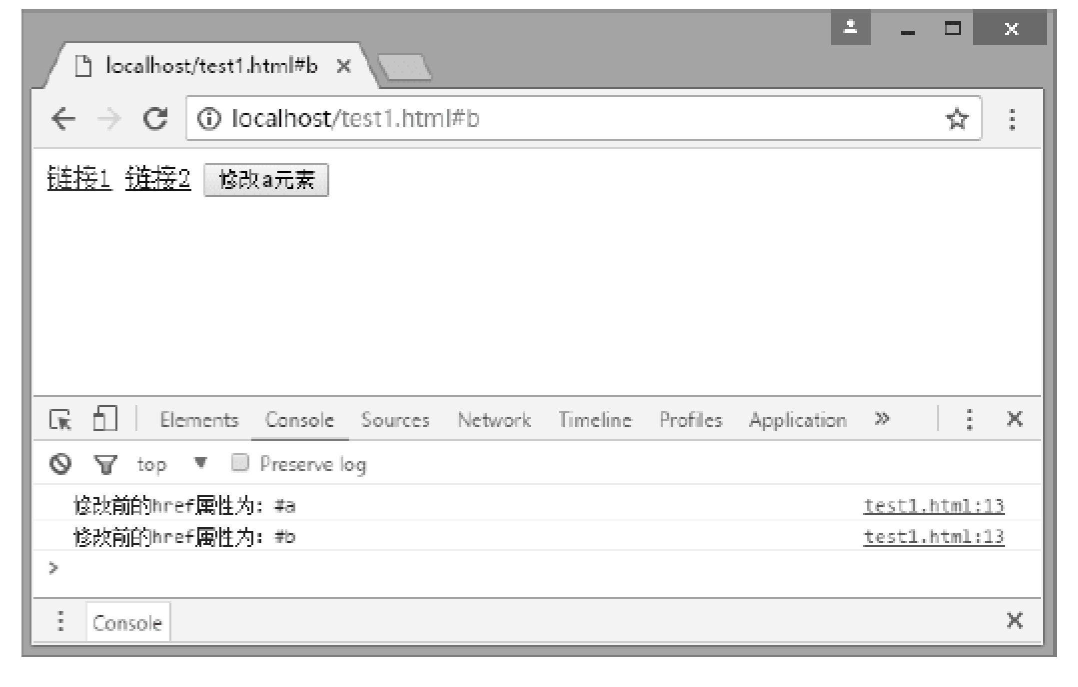


图 18.5　观察 DOM 属性变化



示例效果

示例代码如下所示：

```
<a id="a1" href="#a">链接 1</a><a id="a2" href="#b">链接 2</a>
<input type="button" value="修改 a 元素" onclick="changeA();">
<script type="text/javascript">
function onchange(mutationRecords,mutationObserver) {
    for(var i=0;i<mutationRecords.length;i++)
        console.log("修改前的"+mutationRecords[i].attributeName+"属性为："+mutationRecords[i].oldValue);
}
var a1El = document.getElementById('a1');
var a2El = document.getElementById('a2');
var attr= ["href"];
var mo = new window.MutationObserver(onchange),
options = {attributes: true,attributeFilter: attr,attributeOldValue:true};
mo.observe(a1El,options);
mo.observe(a2El,options);
function changeA(){
    a1El.setAttribute("href","http://www.baidu.com");
    a2El.setAttribute("href","http://www.weibo.com");
}
</script>
```

18.4　在 线 练 习

练习 HTML5 其他 API。



在线练习

第19章 CSS3基础

CSS3 在 CSS 2.1 基础上新增了很多强大功能，如圆角、阴影、多图背景、渐变背景、弹性布局、变形、动画、设备响应等。本章将简单介绍 CSS3 的基本状况，同时详细讲解 CSS3 新增的选择器，如果想设计一个干净的、轻量级的标签，并使结构与表现更好地分离，CSS3 选择器是非常有用的。它可以让设计师更方便地维护样式表，减少对 HTML 类名和 ID 名的依赖，以及对 HTML 结构的依赖，使编写代码更加简单轻松。

【学习重点】

- ⏭ 了解 CSS 历史。
- ⏭ 了解 CSS3 基本状况。
- ⏭ 了解 CSS3 选择器。
- ⏭ 正确使用属性选择器。
- ⏭ 灵活使用伪类选择器进行网页设计。



19.1　CSS3 概述

1996 年 12 月，CSS 的第一个版本被正式发布（http://www.w3.org/TR/CSS1/）；1998 年 5 月，CSS 2.0 版本正式发布（http://www.w3.org/TR/CSS2/）。CSS3 的开发工作在 2000 年之前就已经开始，但各方博弈时间太久，2002 年 W3C 启动了 CSS 2.1 的开发，这是 CSS 2.0 的修订版，它纠正 CSS 2.0 版本中的一些缺陷，更精确地描述 CSS 的浏览器实现，2004 年 CSS 2.1 正式发布，到 2006 年年底得到完善，它成为浏览器支持最完整的版本。为了方便各主流浏览器根据需要渐进式支持，CSS3 按模块化进行全新设计，这些模块可以独立发布和实现，这也为日后 CSS 的扩展奠定了基础。

19.1.1　CSS3 模块

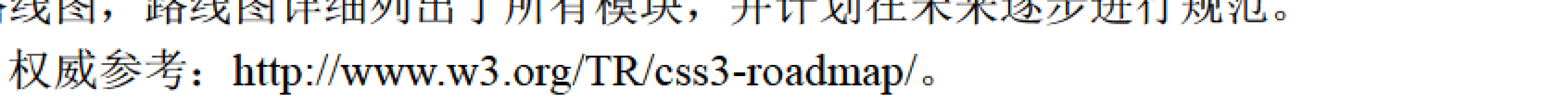


线上阅读

CSS 1 和 CSS 2.1 都是单一的规范，其中 CSS 1 主要定义了网页对象的基本样式，如字体、颜色、背景、边框等。CSS 2 添加了高级概念：浮动、定位，以及高级选择器，如子选择器、相邻选择器和通用选择器等。

CSS3 被划分成多个模块组，每个模块组都有自己的规范。这样的好处是整个 CSS3 的规范发布不会因为部分存在争论的部分而影响其他模块的推进。对于浏览器来说，可以根据需要，决定哪些 CSS 功能被支持。对于 W3C 制定者来说，可以根据需要进行针对性的更新，这样更容易扩展新的技术特性。

2001 年 5 月 23 日，W3C 完成了 CSS3 的工作草案，在该草案中制订了 CSS3 发展路线图，路线图详细列出了所有模块，并计划在未来逐步进行规范。



权威参考：http://www.w3.org/TR/css3-roadmap/。



权威参考

要快速了解各主要模块内容和参考地址，请扫码查看。

> 提示：更详细的信息可参见 http://www.w3.org/Style/CSS/current-work.html，其中介绍了 CSS3 具体划分为多少个模块组、CSS3 所有模块组目前所处的状态，以及将在什么时候发布。

19.1.2　CSS3 开发状态

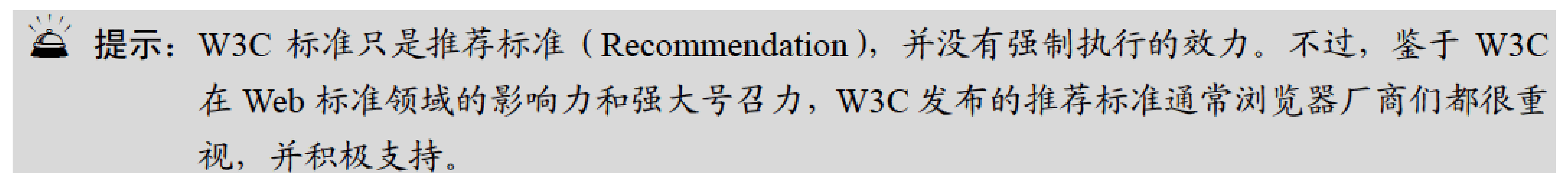
CSS3 每一个模块都有独立的开发和更新计划，进度表如图 19.1 所示，从该图可以看到 CSS3 当前发展的详细进度。

权威参考：http://www.w3.org/Style/CSS/current-work.html。

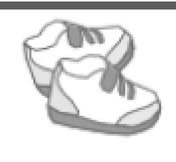
权威参考

其中 Current 列表示模块当前的状态，Upcoming 列表示即将进行的状态。各种状态缩写词说明如下：

- ☑ WD：Working Draft，表示工作草案。
- ☑ LC：Last Call，表示最终工作草案。
- ☑ CR：Candidate Recommendation，表示候选推荐标准。
- ☑ PR：Proposed Recommendation，表示建议推荐标准。
- ☑ REC：Recommendation，表示推荐标准。



> 提示：W3C 标准只是推荐标准（Recommendation），并没有强制执行的效力。不过，鉴于 W3C 在 Web 标准领域的影响力和强大号召力，W3C 发布的推荐标准通常浏览器厂商们都很重视，并积极支持。

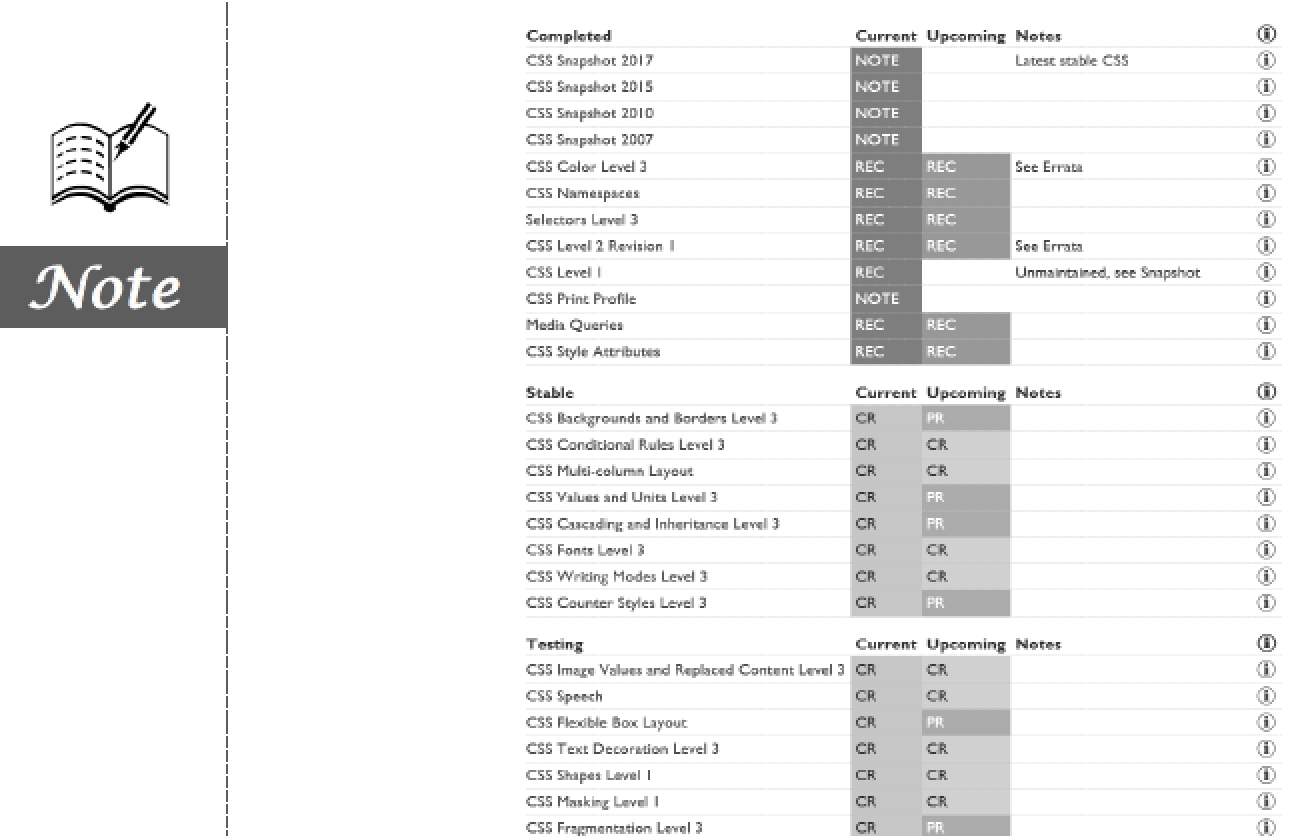


Note

要了解 W3C 标准制作流程，请扫码查看。



线上阅读

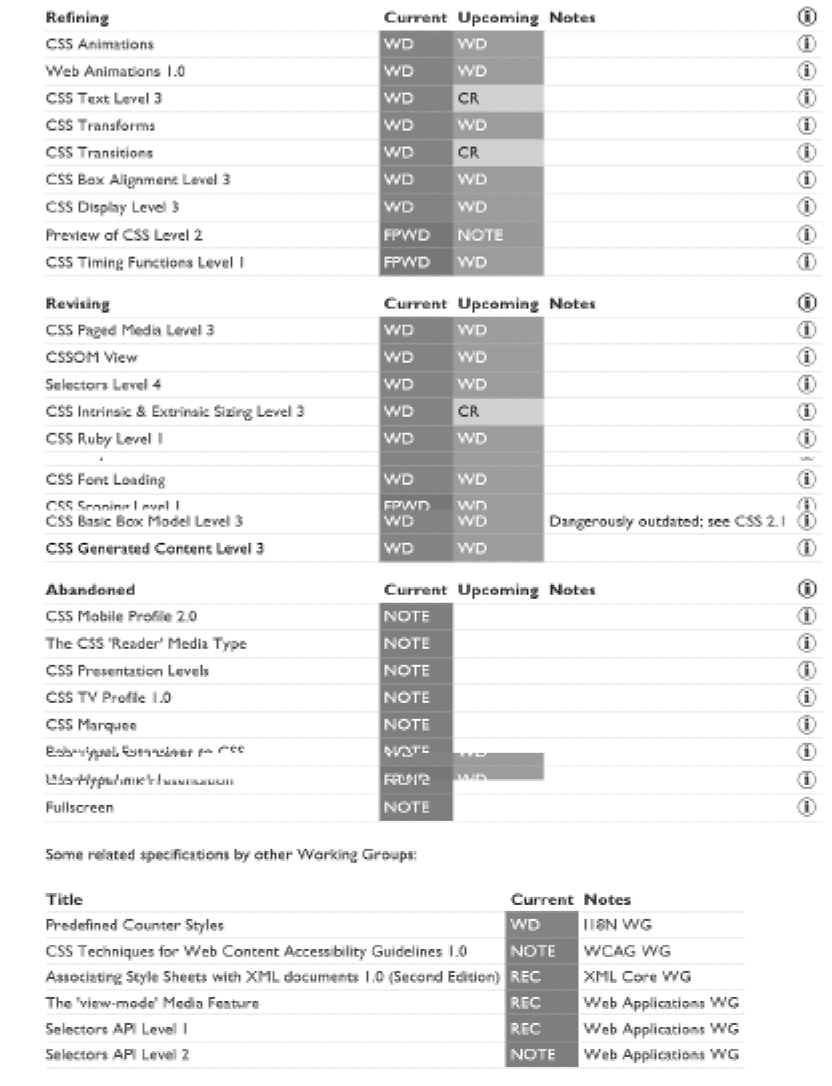


Ordered from most to least stable:

Completed	Current	Upcoming	Notes
CSS Snapshot 2017	NOTE		Latest stable CSS
CSS Snapshot 2015	NOTE		
CSS Snapshot 2010	NOTE		
CSS Snapshot 2007	NOTE		
CSS Color Level 3	REC	REC	See Errata
CSS Namespaces	REC	REC	
Selectors Level 3	REC	REC	
CSS Level 2 Revision 1	REC	REC	See Errata
CSS Level 1	REC		Unmaintained, see Snapshot
CSS Print Profile	NOTE		
Media Queries	REC	REC	
CSS Style Attributes	REC	REC	

Stable	Current	Upcoming	Notes
CSS Backgrounds and Borders Level 3	CR	PR	
CSS Conditional Rules Level 3	CR	CR	
CSS Multi-column Layout	CR	CR	
CSS Values and Units Level 3	CR	PR	
CSS Cascading and Inheritance Level 3	CR	PR	
CSS Fonts Level 3	CR	CR	
CSS Writing Modes Level 3	CR	CR	
CSS Counter Styles Level 3	CR	PR	

Testing	Current	Upcoming	Notes
CSS Image Values and Replaced Content Level 3	CR	CR	
CSS Speech	CR	CR	
CSS Flexible Box Layout	CR	PR	
CSS Text Decoration Level 3	CR	CR	
CSS Shapes Level 1	CR	CR	
CSS Masking Level 1	CR	CR	
CSS Fragmentation Level 3	CR	PR	
CSS Cascading Variables	CR	PR	
Compositing and Blending Level 1	CR	CR	
CSS Syntax Level 3	CR	CR	
CSS Grid Layout Level 1	CR	PR	
CSS Basic User Interface Level 3	CR	PR	
CSS Will Change Level 1	CR	PR	
Media Queries level 4	CR	PR	
Geometry Interfaces Level 1	CR	CR	
CSS Cascading and Inheritance Level 4	CR	PR	
CSS Scroll Snap Level 1	CR	PR	
CSS Containment Level 1	CR	PR	



Refining	Current	Upcoming	Notes
CSS Animations	WD	WD	
Web Animations 1.0	WD	WD	
CSS Text Level 3	WD	CR	
CSS Transforms	WD	WD	
CSS Transitions	WD	CR	
CSS Box Alignment Level 3	WD	WD	
CSS Display Level 3	WD	WD	
Preview of CSS Level 2	FPWD	NOTE	
CSS Timing Functions Level 1	FPWD	WD	

Revising	Current	Upcoming	Notes
CSS Paged Media Level 3	WD	WD	
CSSOM View	WD	WD	
Selectors Level 4	WD	WD	
CSS Intrinsic & Extrinsic Sizing Level 3	WD	CR	
CSS Ruby Level 1	WD	WD	
CSS Font Loading	WD	WD	
CSS Scoping Level 1	FPWD	WD	
CSS Basic Box Model Level 3	WD	WD	Dangerously outdated; see CSS 2.1
CSS Generated Content Level 3	WD	WD	

Abandoned	Current	Upcoming	Notes
CSS Mobile Profile 2.0	NOTE		
The CSS 'Reader' Media Type	NOTE		
CSS Presentation Levels	NOTE		
CSS TV Profile 1.0	NOTE		
CSS Marquee	NOTE		
[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	
Fullscreen	NOTE		

Some related specifications by other Working Groups:

Title	Current	Notes
Predefined Counter Styles	WD	I18N WG
CSS Techniques for Web Content Accessibility Guidelines 1.0	NOTE	WCAG WG
Associating Style Sheets with XML documents 1.0 (Second Edition)	REC	XML Core WG
The 'view-mode' Media Feature	REC	Web Applications WG
Selectors API Level 1	REC	Web Applications WG
Selectors API Level 2	NOTE	Web Applications WG

图 19.1　CSS3 所有模块进度表

19.1.3　浏览器支持状态

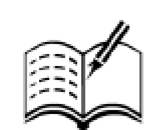
CSS3 特性大部分都已经有了很好的浏览器支持度。各主流浏览器对 CSS3 的支持越来越完善，下面来了解一下两大平台（Mac 和 Windows）、五大浏览器（Chrome、Firefox、Safari、Opera 和 IE）对 CSS3 新特性和 CSS3 选择器的支持情况。

CSS3 属性支持情况如图 19.2 所示（http://fmbip.com/litmus/）。可以看出，完全支持 CSS3 属性的浏览器有 Chrome 和 Safari，而且不管是 Mac 平台还是 Windows 平台都支持。

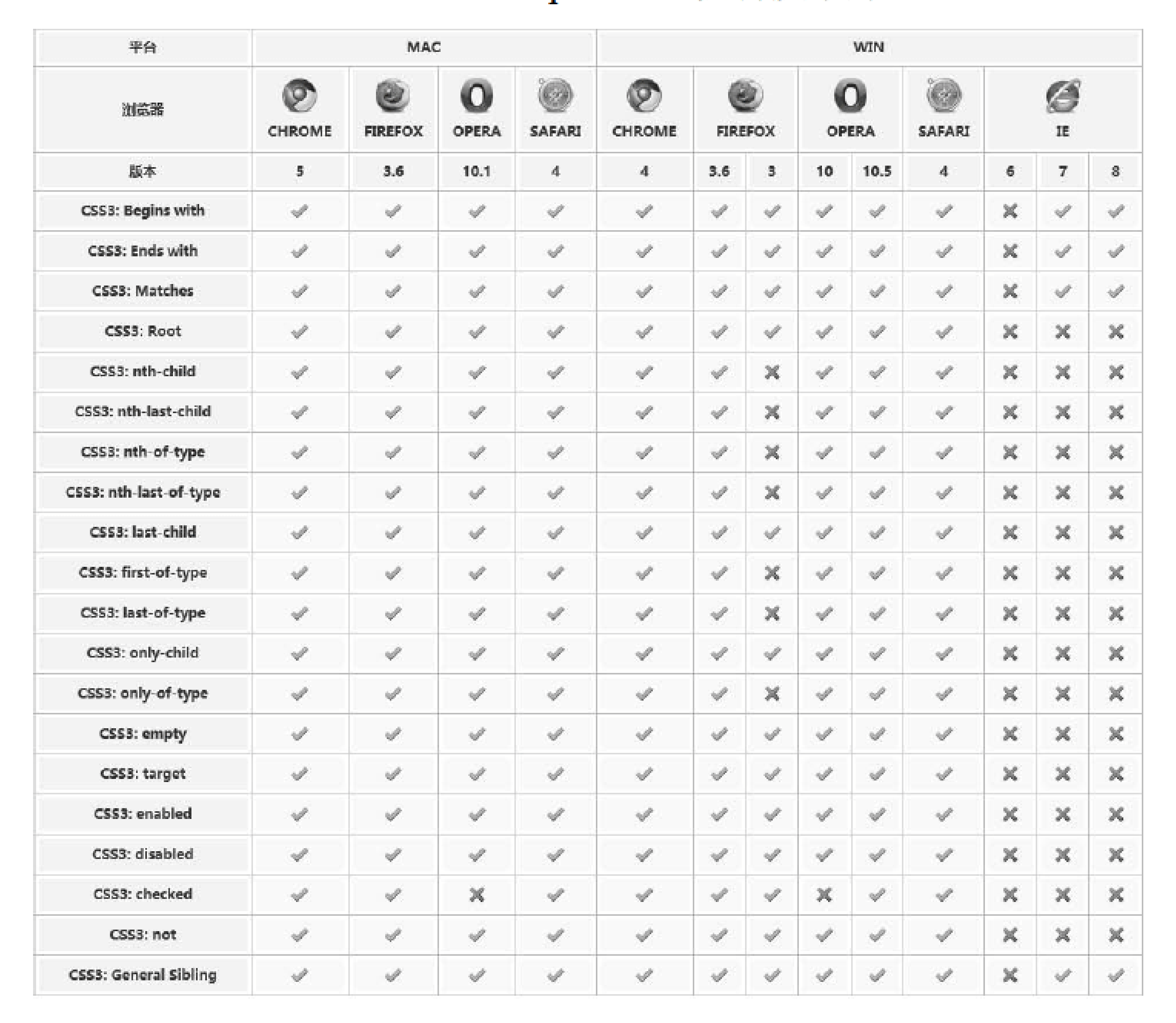
平台	MAC				WIN								
浏览器	CHROME	FIREFOX	OPERA	SAFARI	CHROME	FIREFOX		OPERA		SAFARI	IE		
版本	5	3.6	10.1	4	4	3.6	3	10	10.5	4	6	7	8
RGBA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✘
HSLA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✘
Multiple Backgrounds	✔	✔	✘	✔	✔	✔	✘	✘	✔	✔	✘	✘	✘
Border Image	✔	✔	✘	✔	✔	✔	✘	✘	✔	✔	✘	✘	✘
Border Radius	✔	✔	✘	✔	✔	✔	✔	✘	✔	✔	✘	✘	✘
Box Shadow	✔	✔	✘	✔	✔	✔	✘	✘	✔	✔	✘	✘	✘
Opacity	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✘
CSS Animations	✔	✘	✘	✔	✔	✘	✘	✘	✘	✔	✘	✘	✘
CSS Columns	✔	✔	✘	✔	✔	✔	✔	✘	✘	✔	✘	✘	✘
CSS Gradients	✔	✔	✘	✔	✔	✔	✘	✘	✘	✔	✘	✘	✘
CSS Reflections	✔	✘	✘	✔	✔	✘	✘	✘	✘	✔	✘	✘	✘
CSS Transforms	✔	✔	✘	✔	✔	✔	✘	✘	✔	✔	✘	✘	✘
CSS Transforms 3D	✔	✘	✘	✔	✔	✘	✘	✘	✘	✔	✘	✘	✘
CSS Transitions	✔	✘	✘	✔	✔	✘	✘	✘	✔	✔	✘	✘	✘
CSS FontFace	✔	✔	✔	✔	✔	✔	✘	✔	✔	✔	✔	✔	✔

图 19.2　CSS3 属性支持列表





CSS3 选择器支持情况如图 19.3 所示（http://fmbip.com/litmus/）。除了 IE 家族和 Firefox 3，其他几乎全部支持。Chrome、Safari、Firefox 3.6、Opera 10.5 支持度最好。



平台	MAC				WIN								
浏览器	CHROME	FIREFOX	OPERA	SAFARI	CHROME	FIREFOX		OPERA		SAFARI	IE		
版本	5	3.6	10.1	4	4	3.6	3	10	10.5	4	6	7	8
CSS3: Begins with	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
CSS3: Ends with	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
CSS3: Matches	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
CSS3: Root	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: nth-child	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: nth-last-child	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: nth-of-type	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: nth-last-of-type	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: last-child	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: first-of-type	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: last-of-type	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: only-child	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: only-of-type	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗	✗
CSS3: empty	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: target	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: enabled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: disabled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: checked	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	✗	✗	✗
CSS3: not	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
CSS3: General Sibling	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓


图 19.3　CSS3 选择器支持列表

注意：各主流浏览器也定义大量私有属性，方便用户体验 CSS3 的新特性。例如：

☑ 以 Webkit 引擎为核心的浏览器（如 Safari、Chrome）的私有属性都以-webkit-前缀定义；
☑ 以 Gecko 引擎为核心的浏览器（如 Firefox）的私有属性都以-moz-前缀定义；
☑ 以 Konqueror 引擎为核心的浏览器的私有属性都以-khtml-前缀定义；
☑ Opera 浏览器的私有属性以-o-前缀定义；
☑ Internet Explorer 浏览器的私有属性以-ms-前缀定义，IE 8+支持-ms-前缀。

19.2　CSS3 选择器概述

根据选择器结构的不同，可以把 CSS 选择器分为四大类：

☑ 元素选择器，如表 19.1 所示。
☑ 关系选择器，如表 19.2 所示。
☑ 属性选择器，如表 19.3 所示。
☑ 伪选择器，包括伪类选择器（如表 19.4 所示）和伪对象选择器（如表 19.5 所示）。根据执行任务不同，伪类选择器又分为六种：
  - 动态伪类
  - 目标伪类





权威参考

Note

➢ 语言伪类
➢ 状态伪类
➢ 结构伪类
➢ 否定伪类

选择器模块权威参考：http://www.w3.org/TR/css3-selectors/。

表 19.1　元素选择器列表

选　择　器	说　　明
*	通配选择器，选定所有对象
E	类型选择器，匹配所有同类标签的元素
.className	类选择器，匹配 class 属性值包含 className 的元素。注意，E.className 表示限定元素类选择器
#IDName	ID 选择器，匹配 id 属性值等于 IDName 的元素。注意，E#IDName 表示限定元素 ID 选择器

表 19.2　关系选择器列表

选　择　器	说　　明
E,F	分组选择器，同时匹配 E 和 F 两个子选择器匹配的对象，子选择器之间用逗号分隔
E F	包含选择器，匹配所有被 E 元素包含的 F 元素
E > F	子选择器，匹配 E 元素的所有子元素 F
E + F	相邻选择器，匹配紧贴在 E 元素之后的 F 元素，元素 E 与 F 必须同属一个父级
E ~ F	兄弟选择器，匹配 E 元素后面的所有兄弟元素 F，元素 E 与 F 必须同属一个父级。CSS3 新增

表 19.3　属性选择器列表

选　择　器	说　　明
E[att]	匹配具有 att 属性的 E 元素。注意，E 可以省略，如[cheacked]，以下相同
E[att="val"]	匹配具有 att 属性，且属性值等于 val 的 E 元素
E[att~="val"]	匹配具有 att 属性，且属性值为一用空格分隔的字词列表，其中一个等于 val 的 E 元素。注意，包含只有一个值且该值等于 val 的情况
E[att|="val"]	匹配具有 att 属性，其值是以 val 开头并用连接符“-”分隔的字符串的 E 元素。注意，如果值仅为 val，也将被选择
E[att^="val"]	匹配具有 att 属性，且属性值为以 val 开头的字符串的 E 元素。CSS3 新增
E[att$="val"]	匹配具有 att 属性，且属性值为以 val 结尾的字符串的 E 元素。CSS3 新增
E[att*="val"]	匹配具有 att 属性，且属性值为包含 val 的字符串的 E 元素。CSS3 新增

提示：CSS 支持并列使用多个属性选择器，以匹配同时满足多个选择器，如 blockquote[class=quote][cite] { color:#f00; }。



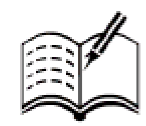


表 19.4　伪类选择器列表

选　择　器	说　　明
E:link	设置超链接 a 在未被访问前的样式
E:visited	设置超链接 a 在其链接地址已被访问过时的样式
E:hover	设置元素在其鼠标悬停时的样式
E:active	设置元素在被用户激活（在鼠标单击与释放之间发生的事件）时的样式
E:focus	设置对象在成为输入焦点时的样式
E:lang(fr)	匹配使用特殊语言的 E 元素
E:not(s)	匹配不含有 s 选择符的元素 E。CSS3 新增
E:root	匹配 E 元素在文档的根元素。在 HTML 中，根元素永远是 HTML。CSS3 新增
E:first-child	匹配父元素的第一个子元素 E。CSS3 新增
E:last-child	匹配父元素的最后一个子元素 E。CSS3 新增
E:only-child	匹配父元素仅有的一个子元素 E。CSS3 新增
E:nth-child(n)	匹配父元素的第 n 个子元素 E，假设该子元素不是 E，则选择符无效。CSS3 新增
E:nth-last-child(n)	匹配父元素的倒数第 n 个子元素 E，假设该子元素不是 E，则选择符无效。CSS3 新增
E:first-of-type	匹配同类型中的第一个同级兄弟元素 E。CSS3 新增
E:last-of-type	匹配同类型中的最后一个同级兄弟元素 E。CSS3 新增
E:only-of-type	匹配同类型中的唯一的一个同级兄弟元素 E。CSS3 新增
E:nth-of-type(n)	匹配同类型中的第 n 个同级兄弟元素 E。CSS3 新增
E:nth-last-of-type(n)	匹配同类型中的倒数第 n 个同级兄弟元素 E。CSS3 新增
E:empty	匹配没有任何子元素（包括 text 节点）的元素 E。CSS3 新增
E:checked	匹配用户界面处于选中状态的元素 E。注意，用于 input 的 type 为 radio 与 checkbox 时。CSS3 新增
E:enabled	匹配用户界面上处于可用状态的元素 E。CSS3 新增
E:disabled	匹配用户界面上处于禁用状态的元素 E。CSS3 新增
E:target	匹配相关 URL 指向的 E 元素。CSS3 新增
@page :first	设置在打印时页面容器第一页使用的样式。注意，仅用于@page 规则
@page :left	设置页面容器位于装订线左边的所有页面使用的样式。注意，仅用于@page 规则
@page :right	设置页面容器位于装订线右边的所有页面使用的样式。注意，仅用于@page 规则

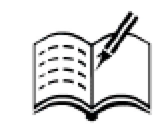
表 19.5　伪对象选择器列表

选　择　器	说　　明
E:first-letter/E::first-letter	设置对象内的第一个字符的样式。注意，仅作用于块对象。CSS3 完善
E:first-line/E::first-line	设置对象内的第一行的样式。注意，仅作用于块对象。CSS3 完善
E:before/E::before	设置在对象前发生的内容。与 content 属性一起使用，且必须定义 content 属性。CSS3 完善
E:after/E::after	设置在对象后发生的内容。与 content 属性一起使用，且必须定义 content 属性。CSS3 完善
E::placeholder	设置对象文字占位符的样式。CSS3 新增
E::selection	设置对象被选择时的样式。CSS3 新增

注意：CSS3 将伪对象选择符前面的单个冒号（:）修改为双冒号（::），用以区别伪类选择符，但以前的写法仍然有效。




19.3　使用CSS3选择器



Note



视频讲解

当把两个或多个简单的选择器组合在一起，就形成了一个复杂的选择器，通过组合选择器可以设计高级选择器，精准匹配页面对象。下面针对CSS3新增选择进行说明和示例演示。

19.3.1　兄弟选择器

兄弟选择器通过波浪符号（~）标识符进行定义，语法格式如下：

```
E~F { sRules }
```

其基本结构是第一个选择器指定同级前置元素，后面的选择器指定其后同级所有匹配元素。前后选择符的关系是兄弟关系，即在HTML结构中两个标签前为兄、后为弟，否则样式无法应用。

【示例】下面是一个简单的示例，具体样式代码如下：

```
<style type="text/css">
/*  相邻选择(E+F) */
h3 + p { color: #00f; font-size:24px; }
/*  兄弟选择(E~F) */
h3 ~ p { color: #f00;    }
</style>
<div class="header">
    <h3>关系选择器</h3>
    <p>E,F：分组选择器，同时匹配 E 和 F 两个子选择器匹配的对象，子选择器之间用逗号分隔。</p>
    <p>E F：包含选择器，匹配所有被 E 元素包含的 F 元素。</p>
    <p>E > F：子选择器，匹配 E 元素的所有子元素 F。</p>
    <p>E + F：相邻选择器，匹配紧贴在 E 元素之后 F 元素，元素 E 与 F 必须同属一个父级。</p>
    <p>E ~ F：兄弟选择器，匹配 E 元素后面的所有兄弟元素 F，元素 E 与 F 必须同属一个父级。CSS3 新增。</p>
</div>
```

兄弟选择器能够选择前置元素后同级的所有匹配元素，而相邻选择器只能选择前置元素后相邻的一个匹配元素。在浏览器中预览，效果如图19.4所示。可以看到第1段文本应用了“color: #f00;”和“font-size:24px;”，后面4段文本应用了“color: #f00;”，说明“h3 + p { color: #00f; font-size:24px; }”仅作用于第一段文本，而“h3 ~ p { color: #f00;　}”作用于所有段落文本。

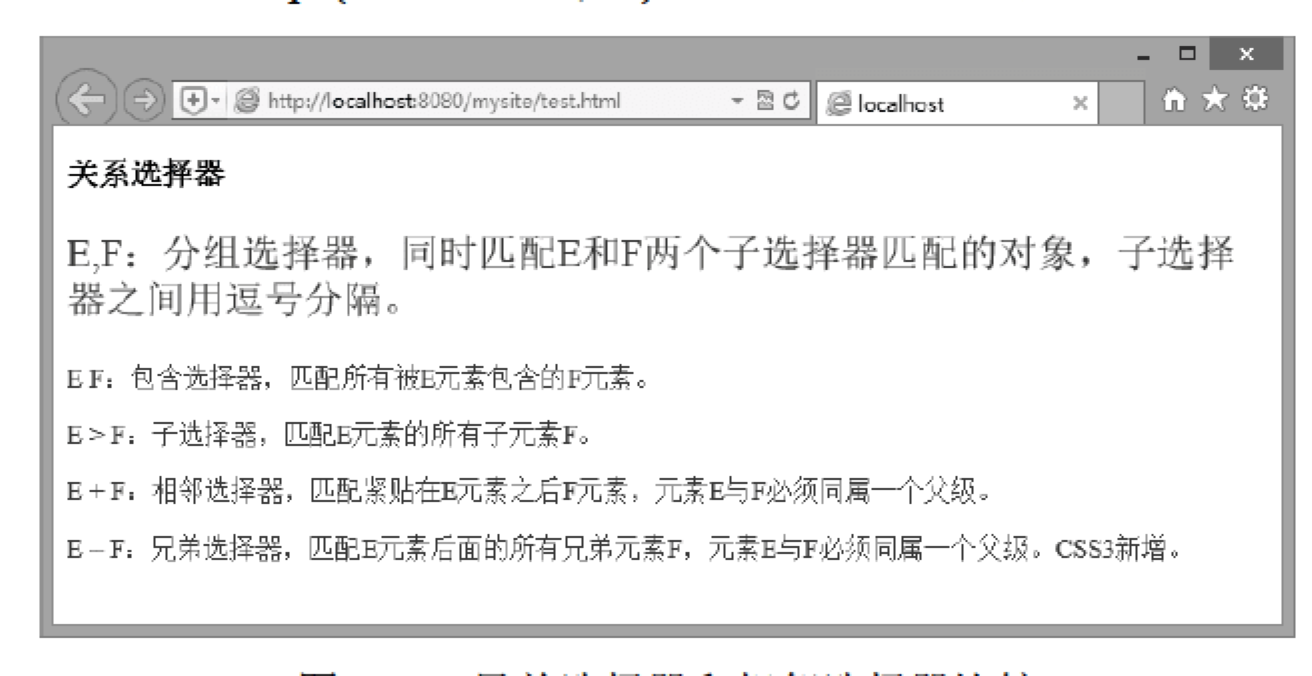
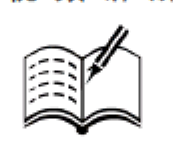


图19.4　兄弟选择器和相邻选择器比较





视 频 讲 解



Note

19.3.2　属性选择器

属性选择器早在 CSS2 中就被引入，如 E[attr]、E[attr="value"]、E[attr~="value"]和 E[attr|="value"]。CSS3 在此基础上新增加了 3 个属性选择器，如 E[attr^="value"]、E[attr$="value"]和 E[attr*="value"]。其与已定义的 4 个属性选择器，构成了强大的 HTML 属性过滤器。

属性选择器以中括号作为语法标识符，在中括号中可以包含 HTML 属性名或者属性值，并通过“^”“$”“|”等运算符定义不同形式的属性选择器，语法格式如下：

```
[属性选择符]
```

CSS3 属性选择器的具体说明可以参考表 19.3，下面结合具体示例进行演示。

【示例 1】下面示例为所有包含 href 属性的超链接定义背景色。

```
<style type="text/css">
a[href] { background-color: #009966;    /*  设置背景色  */ }
</style>
<p><a name="anchor">存在属性 href 才行</a></p>
<p><a href="#">存在属性 href 才行</a></p>
```

【示例 2】下面示例为表单元素类型为文本框（type=text）的元素设置背景色。

```
<style type="text/css">
input[type=text] { background: #CC6633 /*  设置背景色  */ }
</style>
<p><input type="text">属性值为 text 才行</p>
<p><input type="textarea">属性值为 text 才行</p>
```

【示例 3】下面示例为类选择器设置背景色样式。这里 class 属性包含多个值，每个值之间用空格分隔，其中包括 first。

```
<style type="text/css">
[class~=first] { background: #0099FF    /*  设置背景色  */ }
</style>
<ul>
    <li class="first">属性值中存在或者含有 first 需要空格分隔</li>
    <li class="second">属性值中存在或者含有 first 需要空格分隔</li>
    <li class="third">属性值中存在或者含有 first 需要空格分隔</li>
    <li class="first second">属性值中存在或者含有 first 需要空格分隔</li>
    <li class="first third">属性值中存在或者含有 first 需要空格分隔</li>
    <li class="second third">属性值中存在或者含有 first 需要空格分隔</li>
    <li class="first second third">属性值中存在或者含有 first 需要空格分隔</li>
</ul>
```

【示例 4】本例匹配 class 属性值中包含 first 字符串单元，且多个字符串之间用连字符分为匹配的标签设置背景色。

```
<style type="text/css">
[class|="first"] { background-color: #66CC33  /*  设置背景色  */ }
</style>

<!--结构与示例 3 相同-->
```

【示例 5】下面示例为<p>标签中定义提示属性 title，且值以 good 开头设置背景色。




```
<style type="text/css">
p[title^="good"] { background-color: #CC6666       /*  设置背景色  */ }
</style>
<p title="hello">属性的开头必须是 good</p>
<p title="goodmor">属性的开头必须是 good</p>
<p title="Tgoodmor">属性的开头必须是 good</p>
```

Note

【示例 6】下面示例为提示属性 title 的值以 bye 结尾的<p>标签设置背景色。

```
<style type="text/css">
p[title$="bye"] { background-color: #009933   /*  设置背景色  */ }
</style>
<p title="hello">属性中 bye 需要在末尾</p>
<p title="goodbye">属性中 bye 需要在末尾</p>
<p title="goodbye-2">属性中 bye 需要在末尾</p>
```

【示例 7】在下面示例中分别定义了 5 个模糊匹配的属性选择器，然后把匹配的 div 元素显示出来以测试浏览器是否支持该属性选择器，如图 19.5 所示。

```
<style type="text/css">
div { display: none; }/*  隐藏所有 div 元素  */
[class|="blue"] { display: block; }/*  连字符匹配  */
[class~="blue"] { display: block; }/*  空白符匹配  */
[class^="Red"] { display: block; }/*  前缀匹配  */
[class$="Green"] { display: block; }/*  后缀匹配  */
[class*="gre"] {display: block; }/*  子字符串匹配  */
</style>
<div class="red-blue-green">支持[|=]（连字符匹配）属性选择器</div>
<div class="red blue green">支持[~=]（空白符匹配）属性选择器</div>
<div class="Red-blue-green">支持[^=]（前缀匹配）属性选择器</div>
<div class="red-blue-Green">支持[$=]（后缀匹配）  属性选择器</div>
<div class="red-blue-green">支持[*=]:（子字符串匹配）属性选择器</div>
```

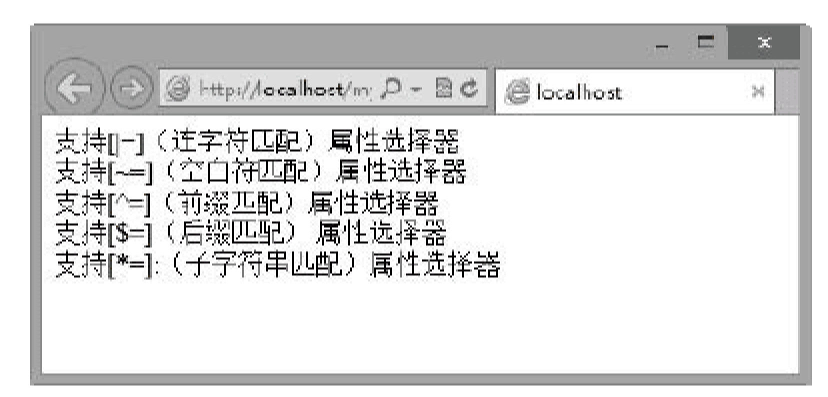


图 19.5　模糊匹配属性选择器演示效果

提示：如果省略了属性选择器的指定标签选择器，这时它将匹配任意标签元素。



视频讲解

19.3.3　伪类选择器

伪类是一种特殊的类选择器，它的用处就是可以对不同状态或行为下的元素定义样式，这些状态或行为是无法通过静态的选择器匹配的，具有动态特性。

伪类选择器以冒号（:）作为语法标识符。冒号前可以添加选择符，限定伪类应用的范围，冒号后为伪类名称，冒号前后没有空格。语法格式如下：

```
:伪类名称
```



CSS 伪类选择器有两种用法方式：

☑　单纯式

```
E:pseudo-class { property:value}
```

其中 E 为元素，pseudo-class 为伪类名称，property 是 CSS 的属性，value 为 CSS 的属性值。例如：

```
a:link {color:red;}
```

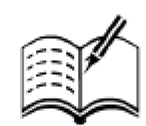
☑　混用式

```
E.class:pseudo-class{property:value}
```

其中.class 表示类选择符。把类选择符与伪类选择符组成一个混合式的选择器，能够设计更复杂的样式，以精准匹配元素。例如：

```
a.selected:hover {color: blue;}
```

在 19.4 节案例实战部分，将会演示多种不同伪类选择器的应用。



19.3.4　伪对象选择器



伪对象选择器主要针对不确定对象定义样式，如第一行文本、第一个字符、前面内容、后面内容。这些对象具体存在，但又无法具体确定，需要使用特定类型的选择器来匹配它们。

伪对象选择器以冒号（:）作为语法标识符。冒号前可以添加选择符，限定伪对象应用的范围，冒号后为伪对象名称，冒号前后没有空格。语法格式如下：

```
:伪对象名称
```

CSS3 新语法格式如下：

```
::伪对象名称
```

提示：伪对象前面包含两个冒号，主要是为了与伪类选择器进行语法区分。

【示例 1】下面示例使用“:before”伪对象选择器在段落文本前面添加三个字符“柳永：”，然后使用“:first-letter”伪对象选择器设置段落文本第一个字符放大显示，定义字体大小为 24 像素，则效果如图 19.6 所示。

```
<style type="text/css">
p:before { content: '柳永：';}
p:first-letter { font-size:24px;}
</style>
<p>衣带渐宽终不悔，为伊消得人憔悴。</p>
```

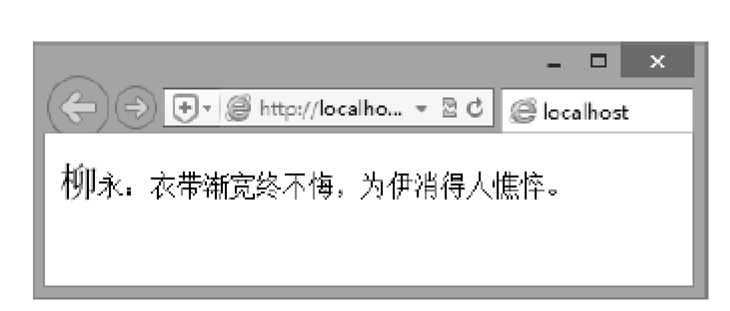
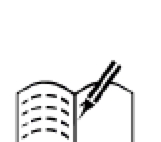


图 19.6　定义第一个字符放大显示

【示例 2】下面示例使用“:first-letter”伪对象选择器设置段落文本第一个字符放大下沉显示，并使用“:first-line”伪对象选择器设置段落文本第一行字符放大带有阴影显示，则效果如图 19.7 所示。

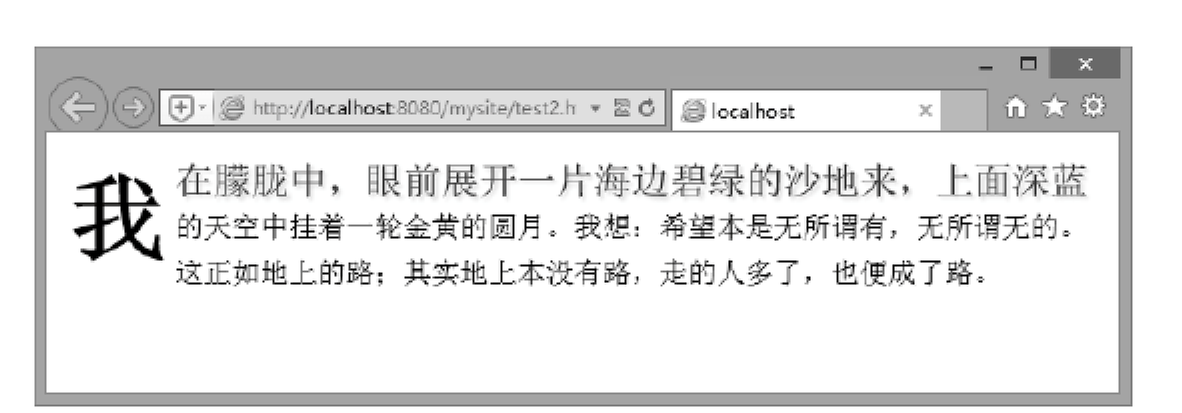




Note

```
<style type="text/css">
p{ font-size:18px; line-height:1.6em;}
p:first-letter {/* 段落文本中第一个字符样式 */
    float:left;
    font-size:60px;
    font-weight:bold;
    margin:26px 6px;
}
p:first-line {/* 段落文本中第一行字符样式 */
    color:red;
    font-size:24px;
    text-shadow:2px 2px 2px rgba(147,251,64,1);
}
</style>

<p>我在朦胧中，眼前展开一片海边碧绿的沙地来，上面深蓝的天空中挂着一轮金黄的圆月。我想：希望本是无所谓有，无所谓无的。这正如地上的路；其实地上本没有路，走的人多了，也便成了路。 </p>
```



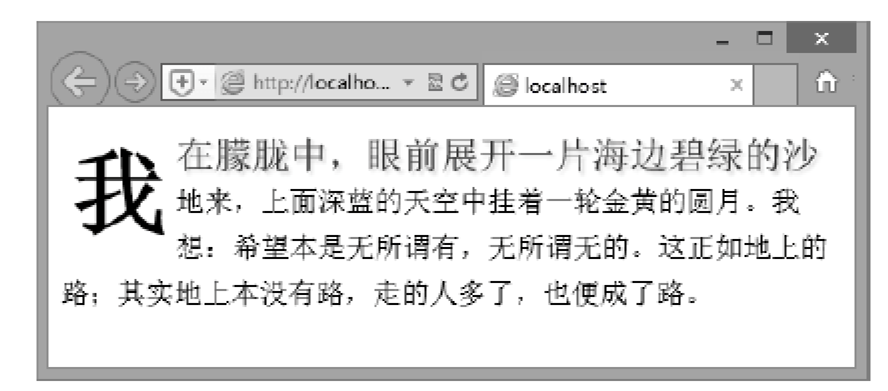


图 19.7　定义第一个字符和第一行字符特殊显示

19.4　案例实战

下面以案例形式练习 CSS3 选择器的应用。

视频讲解

19.4.1　设计按钮样式

动态伪类是一类行为类样式，只有当用户与页面进行交互时有效。包括两种形式：

☑　锚点伪类，如“:link”“:visited”。

☑　行为伪类，如“:hover”“:active”“:focus”。

下面示例将使用动态伪类选择器设计一组 3D 动态效果的按钮样式，效果如图 19.8 所示。



线上阅读



图 19.8　设计 3D 按钮样式

具体操作步骤请扫码学习。





视 频 讲 解

Note

19.4.2　设计列表样式

结构伪类根据文档结构的相互关系来匹配特定的元素，从而减少文档元素的 class 属性和 ID 属性的无序设置，使得文档更加简洁。

结构伪类形式多样，但用法固定，以便设计各种特殊样式效果，结构伪类主要包括下面几种，简单说明如下所示：

- ☑ :first-child：第一个子元素。
- ☑ :last-child：最后一个子元素。
- ☑ :nth-child()：按正序匹配特定子元素。
- ☑ :nth-last-child()：按倒序匹配特定子元素。
- ☑ :nth-of-type()：在同类型中匹配特定子元素。
- ☑ :nth-last-of-type()：按倒序在同类型中匹配特定子元素。
- ☑ :first-of-type：第一个同类型子元素。
- ☑ :last-of-type：最后一个同类型子元素。
- ☑ :only-child：唯一子元素。
- ☑ :only-of-type：同类型的唯一子元素。
- ☑ :empty：空元素。

下面示例设计排行榜栏目列表样式，设计效果如图 19.9 所示。





线 上 阅 读

图 19.9　设计列表样式

具体操作步骤和详细说明请扫码学习。

19.4.3　设计表格样式



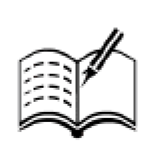
视 频 讲 解

否定选择器只有一个——“:not()”，使用它可排除或者过滤掉特定元素。

在下面示例中设计一个分层表格样式，借助否定伪类选择器和结构伪类选择器，配合 CSS 背景图像设计树形结构标志；借助伪类选择器设计鼠标经过时的动态背景效果，利用 CSS 边框和背景色设计标题行的立体显示效果。演示效果如图 19.10 所示。






Note




线 上 阅 读

图 19.10　设计表格样式

具体操作步骤请扫码学习。


视 频 讲 解

19.4.4　设计表单样式

CSS3 新定义了 3 种常用 UI 状态伪类选择器。简单说明如下：

☑　: enabled

“:enabled”伪类表示匹配指定范围内所有可用 UI 元素。在网页中，UI 元素一般是指包含在 form 元素内的表单元素。例如，在下面表单结构中，input:enabled 选择器将匹配文本框，但不匹配该表单中的按钮。

```
<form>
    <input type="text" />
    <input type="button" disabled="disabled" />
</form>
```

☑　:disabled

“:disabled”伪类表示匹配指定范围内所有不可用 UI 元素。例如，在下面表单结构中，input:disabled 选择器将匹配按钮，但不匹配该表单中的文本框。

```
<form>
    <input type="text" />
    <input type="button" disabled="disabled" />
</form>
```

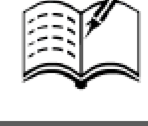
☑　:checked

“:checked”伪类表示匹配指定范围内所有可用 UI 元素。例如，在下面表单结构中，input:checked 选择器将匹配片段中的单选按钮，但不匹配该表单中的复选框。

```
<form>
    <input type="checkbox" />
    <input type="radio" checked="checked" />
</form>
```

在表单中，这些状态伪类是比较常用的，最常见的 type="text"有 enable 和 disabled 两种状态，前者为可写状态，后者为不可写状态。另外，type="radio"和 type="checkbox"有 checked 和 unchecked 两种状态。





Note

【示例】在下面示例中，将设计一个简单的登录表单，为便于观察，同时使用一个不可用的表单对象进行比较。演示效果如图 19.11 所示。在实际应用中，当用户登录完毕，不妨通过脚本把文本框设置为不可用（disabled="disabled"）状态，这时可以通过“:disabled”选择器让文本框显示为灰色，以告诉用户该文本框不可用，这样就不用设计“不可用”样式类，同时把该类添加到 HTML 结构中。

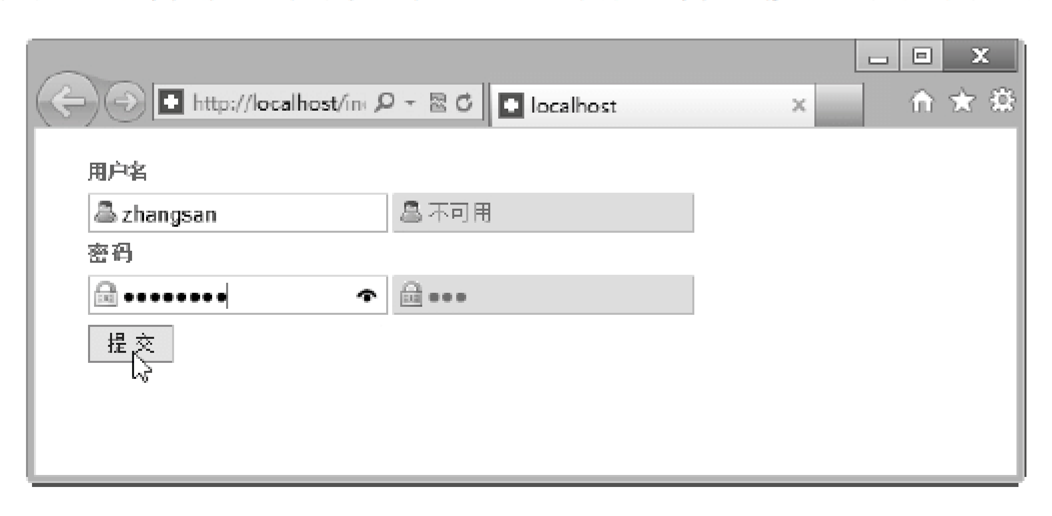


图 19.11　设计登录表单样式



线 上 阅 读

具体操作步骤请扫码学习。

19.4.5　设计锚点样式



视 频 讲 解

目标伪类选择器类型形式如 E:target，它表示选择匹配 E 的所有元素，且匹配元素被相关 URL 指向。该选择器是动态选择器，只有当存在 URL 指向该匹配元素时，样式效果才能够有效。

【示例】下面示例设计当单击页面中的锚点链接跳转到指定标题位置时，该标题会自动高亮显示，以提醒用户当前跳转的位置，效果如图 19.12 所示。

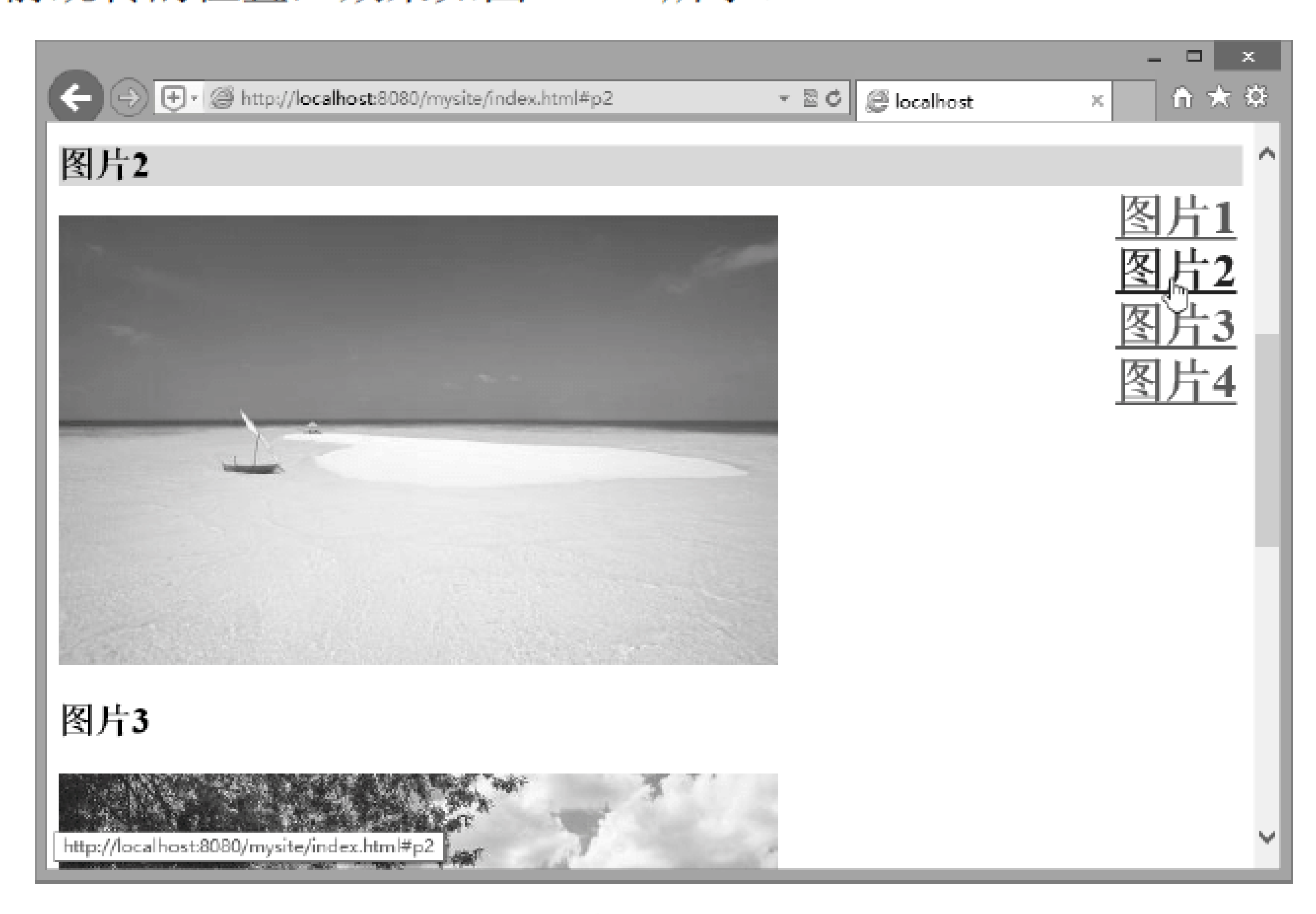


图 19.12　目标伪类样式应用效果



线 上 阅 读

具体代码解析请扫码学习。

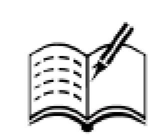
19.4.6　设计超链接样式

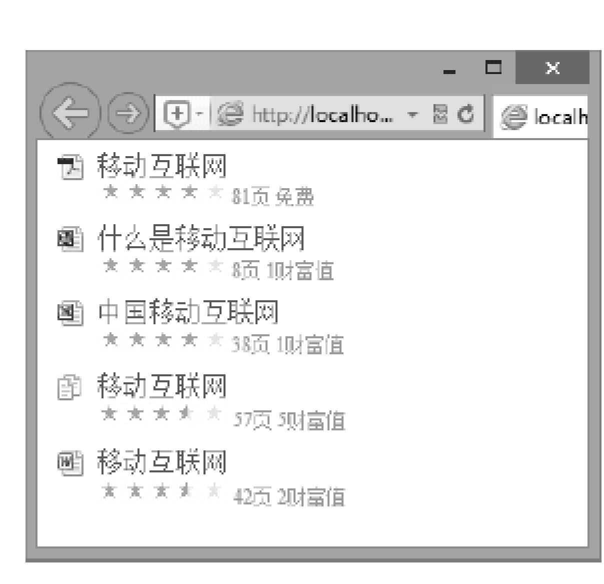


视 频 讲 解

在下面示例中，将模拟百度文库的“相关文档推荐”模块样式设计效果，演示如何利用属性选择器快速并准确匹配文档类型，为不同类型文档超链接定义不同的显示图标，以便浏览者准确识别文档类型。示例演示效果如图 19.13 所示。








线 上 阅 读

图 19.13 设计超链接文档类型的显示图标

具体操作步骤请扫码学习。

19.5 在 线 练 习

灵活使用 CSS，强化基本功训练。


在 线 练 习

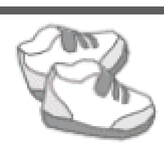
第 20 章

CSS3 文本样式

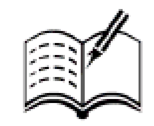
CSS3 优化和增强了 CSS 2.1 的字体和文本属性，使网页文字更具表现力和感染力，丰富了网页文本的样式和版式。用户可以使用网络字体定义特殊的字体类型，摆脱了浏览器所在系统字体的局限；可以选择更多的色彩模式，创建灵活的网页配色体系；通过文本阴影，让字体看起来更美；通过动态内容，让网页内容不再单一，CSS 控制网页内容的显示能力更强。

【学习重点】

- ⏭ 了解 CSS3 文本模块。
- ⏭ 正确处理文本溢出和换行问题。
- ⏭ 能够使用 CSS3 新色彩模式。
- ⏭ 灵活定义文本阴影样式。
- ⏭ 灵活添加动态内容。
- ⏭ 正确使用网络字体。




20.1 CSS3 文本模块



Note

早期 CSS 文本功能就是设置字体、字号、颜色、样式、粗细、间距等。CSS3 的文本功能不再局限于这些基本设置，它优化了已经存在的各个属性，整合了各种不兼容的私有属性，给文本添加一些高级属性，并作为一个独立的模块进行开发，以尽快完善和迭代 CSS 文本功能。

20.1.1 文本模块概述

CSS3 文本模块（TextModule）把与文本相关的属性单独进行规范。文本模块的最早版本是在 2003 年制定的（http://www.w3.org/TR/2003/CR-css3-text-20030514/），2005 年对其进行了修订（http://www.w3.org/TR/2005/WD-css3-text-20050627/），2007 年又进行了系统更新（http://www.w3.org/TR/2007/WD-css3-text-20070306/），最后形成了一个较为完善的文本模型（http://www.w3.org/TR/css3-text/）。

在最终版本的文本模块中，除了新增文本属性外，还对 CSS 2.1 版本中已定义的属性取值进行修补，增加了更多的属性值，以适应复杂环境中文本的呈现。

与 2003 年版本相比，进行了较大的改动，其中主要改动说明如下。

☑ line-break 和 word-break-cjk 属性被 word-break 属性替换。
☑ word-break-inside 属性被 hyphenate 属性替换。
☑ wrap-option 属性被 text-wrap 和 word-break 属性替换。
☑ linefeed-treatment、white-space-treatment 和 all-space-treatment 属性被 white-space-collapse 属性替换。
☑ min-font-size 和 max-font-size 属性被移到下一个 CSS3 版本的字体模块内。
☑ 修改了 text-align 属性中 left 和 right 属性值在垂直文本中的行为。
☑ text-align-last 属性取消了 size 属性值。
☑ text-justify 属性取消了 newspaper 属性值。
☑ word-spacing 和 letter-spacing 属性增加了百分比取值。
☑ text-wrap 属性增加了 suppress 属性值。
☑ 删除 linefeed-treatment 属性。
☑ text-align-last 属性取消了 size 属性值。
☑ text-justify 属性新增了 tibetan 属性值。
☑ punctuation-trim 属性新增加了 end 属性值。
☑ kerning-mode:contextual 被 punctuation-trim:adjacent 替换，其他控制被移至字体模块。
☑ text-shadow 属性现在可以继承。
☑ 新增 text-outline 属性。
☑ 新增 text-emphasis 属性，替换 font-emphasis 属性。
☑ 重新定义了 text-indent 属性。
☑ 重新设计了 hanging-punctuation 属性。

最新版本的文本模型与 2005 年版本相比，也进行了适当修订，其中增加 text-emphasis 和 text-outline 属性，移除了 font-emphasis 属性，其他更多改动细节请参阅官方文档。





Note

20.1.2　文本溢出

text-overflow 属性可以设置超长文本省略显示。基本语法如下所示。

```
text-overflow:clip | ellipsis
```

适用于块状元素，取值简单说明如下：

☑　clip：当内联内容溢出块容器时，将溢出部分裁切掉，为默认值。

☑　ellipsis：当内联内容溢出块容器时，将溢出部分替换为（...）。

> **提示**：在早期 W3C 文档（http://www.w3.org/TR/2003/CR-css3-text-20030514/#textoverflow-mode）中，text-overflow 被纳入规范，但是在最新修订的文档（http://www.w3.org/TR/css3-text/）中没有再包含 text-overflow 属性。

由于 W3C 规范放弃了对 text-overflow 属性的支持，所以，Mozilla 类型浏览器也放弃了对该属性的支持。不过，Mozilla developer center 推荐使用-moz-binding 的 CSS 属性进行兼容。Firefox 浏览器支持 XUL（XUL，一种 XML 的用户界面语言），这样就可以使用-moz-binding 属性来绑定 XUL 里的 ellipsis 属性了。

> **注意**：text-overflow 属性仅是内容注解，当文本溢出时是否显示省略标记，并不具备样式定义的特性。要实现溢出时产生省略号的效果，还应定义两个样式：强制文本在一行内显示（white-space:nowrap）和溢出内容为隐藏（overflow:hidden），只有这样才能实现溢出文本显示省略号的效果。

【示例】下面示例设计新闻列表有序显示，对于超出指定宽度的新闻项，则使用 text-overflow 属性省略并附加省略号，避免新闻换行或者撑开版块，演示效果如图 20.1 所示。



图 20.1　设计固定宽度的新闻栏目

示例代码如下：

```
<style type="text/css">
dl {/*定义新闻栏目外框，设置固定宽度*/
    width:300px;
    border:solid 1px #ccc;
}
dt {/*设计新闻栏目标题行样式*/
    padding:8px 8px;                                    /*增加文本周围空隙*/
    margin-bottom:12px;                                 /*调整底部间距*/
    background:#7FECAD url(images/green.gif) repeat-x;  /*设计标题栏背景图*/
    /*定义字体样式*/
```



```
        font-size:13px; font-weight:bold; color:#71790C;
        text-align:left;                          /*恢复文本默认左对齐*/
        border-bottom:solid 1px #efefef;          /*定义浅色边框线*/
    }
    dd {/*设新闻列表项样式*/
        font-size:0.78em;
        /*固定每个列表项的大小*/
        height:1.5em;width:280px;
        /*为添加新闻项目符号腾出空间*/
        padding:2px 2px 2px 18px;
        /*以背景方式添加项目符号*/
        background: url(images/icon.gif) no-repeat 6px 25%;
        margin:2px 0;
        /*为应用 text-overflow 做准备，禁止换行*/
        white-space: nowrap;
        /*为应用 text-overflow 做准备，禁止文本溢出显示*/
        overflow: hidden;
        -o-text-overflow: ellipsis;               /* 兼容 Opera */
        text-overflow: ellipsis;                  /* 兼容 IE, Safari (WebKit) */
        -moz-binding: url('images/ellipsis.xml#ellipsis');    /* 兼容 Firefox */}
</style>
<dl>
    <dt>唐诗名句精选</dt>
    <dd>海内存知己，天涯若比邻。唐·王勃《送杜少府之任蜀州》 </dd>
    <dd>不知细叶谁裁出，二月春风似剪刀。唐·贺知章《咏柳》 </dd>
    <dd>欲穷千里目，更上一层楼。唐·王之涣《登鹳雀楼》 </dd>
    <dd>野旷天低树，江清月近人。唐·孟浩然《宿建德江》 </dd>
    <dd>大漠孤烟直，长河落日圆。唐·王维《使至塞上》 </dd>
</dl>
```

Note



视频讲解

20.1.3 文本换行

在 CSS3 中，使用 word-break 属性可以定义文本自动换行。基本语法如下所示。

```
word-break:normal | keep-all | break-all
```

取值简单说明如下：

☑ normal：为默认值，依照亚洲语言和非亚洲语言的文本规则，允许在字内换行。

☑ keep-all：对于中文、韩文、日文不允许字断开。适合包含少量亚洲文本的非亚洲文本。

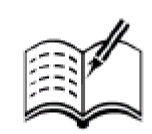
☑ break-all：与 normal 相同，允许非亚洲语言文本行的任意字内断开。该值适合包含一些非亚洲文本的亚洲文本，如使连续的英文字母间断行。

word-wrap 属性没有被广泛支持，特别是 Firefox 和 Opera 浏览器对其的支持比较消极，这是因为早期的 W3C 文本模型（http://www.w3.org/TR/2003/CR-css3-text-20030514/）放弃了对其的支持，而是定义了 wrap-option 属性代替 word-wrap 属性。但是在最新的文本模式（http://www.w3.org/TR/css3-text/）中继续支持该属性，并重定义了属性值。

【补充】

word-break 原来是 IE 私有属性，在 CSS3 中被 Text 模块采用，得到 Chrome 和 Safari 等浏览器的支持，但不支持 keep-all 取值。





另外，IE 自定义了多个换行处理属性：line-break、word-wrap、word-break，CSS 1 也定义了 white-space。这几个属性简单比较如下：

☑ line-break：指定如何（或是否）断行。除了 Firefox，其他浏览器都支持。取值说明如下所示。
  ➢ auto：使用默认的断行规则分解文本。
  ➢ loose：使用最松散的断行规则分解文本，一般用于短行的情况，如报纸。
  ➢ normal：使用最一般的断行规则分解文本。
  ➢ strict：使用最严格的断行原则分解文本。
☑ word-wrap：允许长单词或 URL 地址换行到下一行。所有浏览器都支持。取值说明如下所示。
  ➢ normal：只在允许的断字点换行（浏览器保持默认处理）。
  ➢ break-word：在长单词或 URL 地址内部进行换行。
☑ word-break：指定怎样在单词内断行。取值说明参考上面语法。
☑ white-space：设置如何处理元素中的空格。所有浏览器都支持。取值说明如下所示。
  ➢ normal：默认处理方式。
  ➢ pre：使用等宽字体显示预先格式化的文本，不合并文字间的空白距离，当文字超出边界时不换行。
  ➢ nowrap：强制在同一行内显示所有文本，合并文本间的多余空白。
  ➢ pre-wrap：使用等宽字体显示预先格式化的文本，不合并文字间的空白距离，当文字碰到边界时发生换行。
  ➢ pre-line：保持文本换行，不保留文字间的空白距离，当文字碰到边界时发生换行。

【拓展】

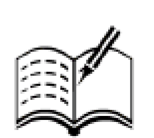
在 IE 浏览器下，使用“word-wrap:break-word;”声明可以确保所有文本正常显示。在 Firefox 浏览器下，中文不会出任何问题。英文语句也不会出问题。但是，长串英文会出问题。为了解决长串英文，一般“word-wrap:break-word;”和“word-break:break-all;”声明结合使用。但是，这种方法会导致普通的英文语句中的单词被断开显示（IE 下也是）。现在的问题主要是长串英文和英文单词被断开的问题。

为了解决这个问题，可使用“word-wrap:break-word;overflow:hidden;”，而不是“word-wrap:break-word;word-break:break-all;”。“word-wrap:break-word;overflow:auto;”在 IE 下没有任何问题，但是在 Firefox 下，长串英文单词就会被遮住部分内容。

【示例】下面示例在页面中插入一个表格，由于标题行文字较多，标题行常被撑开，影响浏览体验。为了解决这个问题，借助 CSS 换行属性进行处理，比较效果如图 20.2 所示。

```
<style type="text/css">
table {
    width: 100%;
    font-size: 14px;
    border-collapse: collapse;              /*定义细线表格*/
    border: 1px solid #cad9ea;              /*添加淡色细线边框*/
    table-layout: fixed;                    /*定义表格在浏览器端逐步解析呈现，避免破坏布局*/
}
th {
```





Note

```
        background-image: url(images/th_bg1.gif);    /*使用背景图模拟渐变背景*/
        background-repeat: repeat-x;                 /*定义背景图平铺方式*/
        height: 30px;
        vertical-align:middle;                       /*垂直居中显示*/
        border: 1px solid #cad9ea;                   /*添加淡色细线边框*/
        padding: 0 1em 0;
        overflow: hidden;                            /*超出范围隐藏显示，避免撑开单元格*/
        word-break: keep-all;                        /*禁止词断开显示*/
        white-space: nowrap;                         /*强迫在一行内显示*/
    }
    td {
        height: 20px;
        border: 1px solid #cad9ea;                   /*添加淡色细线边框*/
        padding: 6px 1em;                            /*增加单元格空隙，避免文本挤在一起*/
    }
    tr:nth-child(even) { background-color: #f5fafe; }
    .w4 { width: 4em; }
    </style>
    <table>
        <tr>
            <th class="w4">与文本换行相关的属性</th> <th>使用说明</th>
        </tr>
        <tr>
            <td>line-break</td><td>......</td>
        </tr>
        <tr>
            <td>word-wrap</td> <td>......</td>
        </tr>
        <tr>
            <td>word-break</td><td>......</td>
        </tr>
        <tr>
            <td>white-space</td><td>......</td>
        </tr>
    </table>
```



与文本换行相关的属性	使用说明
line-break	用于指定如何（或是否）断行。除了Firefox，其它浏览器都支持。取值包括： auto，使用缺省的断行规则分解文本； loose，使用最松散的断行规则分解文本，一般用于短行的情况，如报纸； normal，使用最一般的断行规则分解文本； strict，使用最严格的断行原则分解文本。
word-wrap	允许长单词或URL地址换行到下一行。所有浏览器都支持。取值包括： normal，只在允许的断字点换行（浏览器保持默认处理）； break-word，在长单词或 URL 地址内部进行换行。
word-break	定义文本自动换行。Chrome和Safari浏览器支持不够友好。取值包括： normal：为默认值，允许在字内换行; keep-all,对于中文、韩文、日文，不允许字断开； break-all，与normal相同，允许非亚洲语言文本行的任意字内断开。
white-space	设置如何处理元素中的空格。所有浏览器都支持。取值包括： normal，默认处理方式； pre，显示预先格式化的文本，当文字超出边界时不换行； nowrap，强制在同一行内显示所有文本，合并文本间的多余空白，直到文本结束或者遭遇br对象； pre-wrap，显示预先格式化的文本，不合并文字间的空白距离，当文字碰到边界时发生换行； pre-line，保持文本的换行，不保留文字间的空白距离，当文字碰到边界时发生换行。

（a）处理前



与文本换行	使用说明
line-break	用于指定如何（或是否）断行。除了Firefox，其它浏览器都支持。取值包括： auto，使用缺省的断行规则分解文本； loose，使用最松散的断行规则分解文本，一般用于短行的情况，如报纸； normal，使用最一般的断行规则分解文本； strict，使用最严格的断行原则分解文本。
word-wrap	允许长单词或URL地址换行到下一行。所有浏览器都支持。取值包括： normal，只在允许的断字点换行（浏览器保持默认处理）； break-word，在长单词或 URL 地址内部进行换行。
word-break	定义文本自动换行。Chrome和Safari浏览器支持不够友好。取值包括： normal：为默认值，允许在字内换行; keep-all,对于中文、韩文、日文，不允许字断开； break-all，与normal相同，允许非亚洲语言文本行的任意字内断开。
white-space	设置如何处理元素中的空格。所有浏览器都支持。取值包括： normal，默认处理方式； pre，显示预先格式化的文本，当文字超出边界时不换行； nowrap，强制在同一行内显示所有文本，合并文本间的多余空白，直到文本结束或者遭遇br对象； pre-wrap，显示预先格式化的文本，不合并文字间的空白距离，当文字碰到边界时发生换行； pre-line，保持文本的换行，不保留文字间的空白距离，当文字碰到边界时发生换行。

（b）处理后

图 20.2　禁止表格标题文本换行显示





视 频 讲 解



Note

20.1.4　书写模式

CSS3 增强了文本布局中的书写模式，在 CSS 2.1 定义的 direction 和 unicode-bidi 属性基础上，新增 writing-mode 属性。基本语法如下所示。

```
writing-mode:horizontal-tb | vertical-rl | vertical-lr | lr-tb | tb-rl
```

取值简单说明如下：

☑　horizontal-tb：水平方向自上而下的书写方式，类似 IE 私有值 lr-tb。

☑　vertical-rl：垂直方向自右而左的书写方式，类似 IE 私有值 tb-rl。

☑　vertical-lr：垂直方向自左而右的书写方式。

☑　lr-tb：左-右，上-下。对象中的内容在水平方向上从左向右流入，后一行在前一行的下面显示。

☑　tb-rl：上-下，右-左。对象中的内容在垂直方向上从上向下流入，自右向左。后一竖行在前一竖行的左面。全角字符是竖直向上的，半角字符如拉丁字母或片假名顺时针旋转 90°。



权 威 参 考

权威参考：http://www.w3.org/TR/2011/WD-css3-writing-modes-20110901/。

【补充】

direction 设置文本流方向，取值包括：ltr，文本流从左到右；rtl，文本流从右到左。unicode-bidi 用于在同一个页面内显示不同方向的文本，与 direction 属性一起使用。

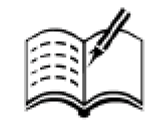
【示例 1】下面示例设计唐诗从右侧流入，自上而下显示，效果如图 20.3 所示。

```
<style type="text/css">
#box {
    float: right;
    writing-mode: tb-rl;
    -webkit-writing-mode: vertical-rl;
    writing-mode: vertical-rl;
}
</style>
<div id="box">
    <h2>春晓</h2>
    <p>春眠不觉晓，处处闻啼鸟。夜来风雨声，花落知多少。</p>
</div>
```

【示例 2】配合 margin-top: auto 和 margin-bottom: auto 声明，可以设计栏目垂直居中效果，如图 20.4 所示。

```
<style type="text/css">
.box {
    width: 400px; height: 300px;
    background-color: #f0f3f9;
    writing-mode: tb-rl;
    -webkit-writing-mode: vertical-rl;
    writing-mode: vertical-rl;
}
.auto {
    margin-top: auto;                /*垂直居中*/
```





Note

```
        margin-bottom: auto;                    /*垂直居中*/
        height:120px;
    }
    img { height:120px;}
    </style>

    <div class="box">
        <div class="auto"><img src="images/bg.png"></div>
    </div>
```





图 20.3　设计唐诗传统书写方式

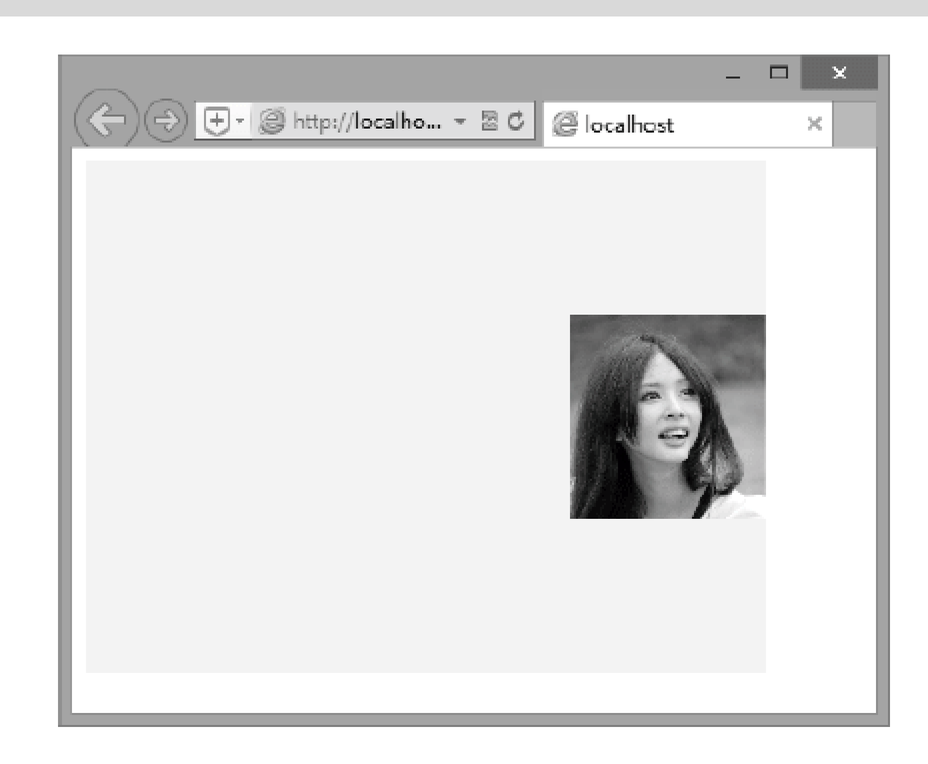


图 20.4　设计垂直居中布局

【示例 3】下面示例设计一个象棋棋子，然后定义当超链接被激活时，首行文本缩进 4 个像素。由于使用了垂直书写模式，则文本向下移动 4 个像素，这样就可以模拟一种动态下沉效果，如图 20.5 所示。

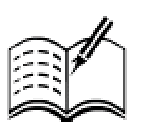
```
    <style type="text/css">
    .btn {
        width: 80px; height: 80px;              /*固定大小*/
        line-height: 80px;                      /*垂直居中*/
        font-size: 62px;                        /*大字体*/
        cursor: pointer;                        /*手形指针样式*/
        text-align: center;                     /*文本居中显示*/
        text-decoration:none;                   /*清除下划线*/
        color: #a78252;                         /*字体颜色*/
        background-color: #ddc390;              /*增加背景色*/
        border: 6px solid #ddc390;              /*增加粗边框*/
        border-radius: 50%;                     /*定义圆形显示*/
        /*定义阴影和内阴影边线*/
        box-shadow: inset 0 0 0 1px #d6b681, 0 1px, 0 2px, 0 3px, 0 4px;
        writing-mode: tb-rl;
        -webkit-writing-mode: vertical-rl;
        writing-mode: vertical-rl;
    }
    .btn:active { text-indent: 4px;}
    </style>

    <a href="#" class="btn">将</a>
```





图 20.5　设计首字下沉特效





20.1.5　initial 值

initial 表示初始化属性的值，所有的属性都可以接受该值。如果想重置某个属性为浏览器默认设置，那么就可以使用该值，这样就可以取消用户定义的 CSS 样式。

注意：IE 暂不支持该属性值。

【示例】在下面示例中，页面中插入了四段文本，然后在内部样式表中定义这四段文本蓝色、加粗显示，字体大小为 24 像素，显示效果如图 20.6 所示。

```
<style type="text/css">
p {
    color: blue;
    font-size:24px;
    font-weight:bold;
}
</style>
<p>春眠不觉晓，</p>
<p>处处闻啼鸟。</p>
<p>夜来风雨声，</p>
<p>花落知多少。</p>
```

如果想禁止第一句和第三句用户定义的样式，只需在内部样式表中添加一个独立样式，然后把文本样式的值都设为 initial 值就可以了，具体代码如下所示，运行结果如图 20.7 所示。

```
p:nth-child(odd){
    color: initial;
    font-size:initial;
    font-weight:initial;
}
```

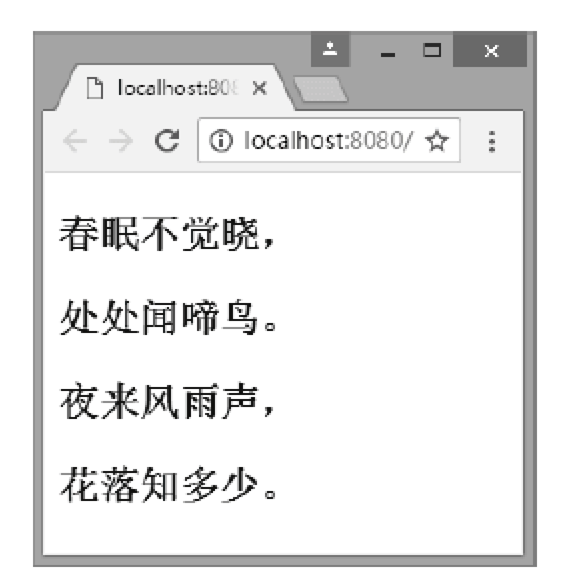


图 20.6　定义段落文本样式

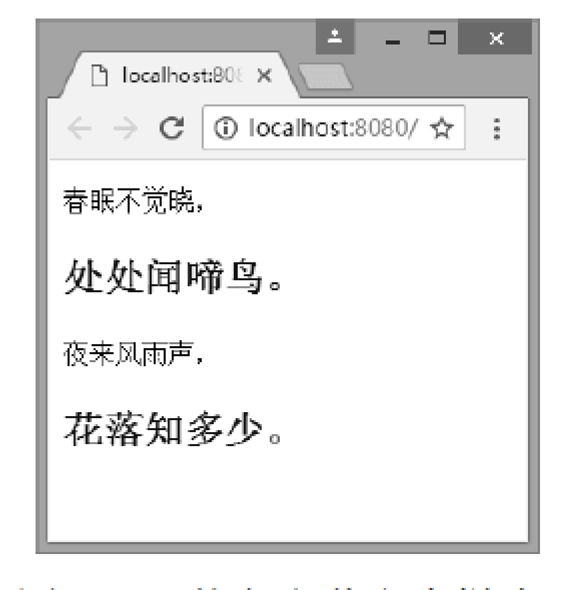
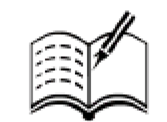


图 20.7　恢复段落文本样式



在浏览器中可以看到，第一句和第三句文本恢复为默认的黑色、常规字体，大小为 16 像素。

20.1.6　inherit 值



Note



视 频 讲 解

inherit 表示属性能够继承祖先的设置值，所有的属性都可以接受该值。

【示例】下面示例设置一个包含框，高度为 200 像素，包含两个盒子，定义盒子高度分别为 100%和 inherit，正常情况下都会显示 200 像素，但是在特定情况下，如当盒子被定义为绝对定位显示，则设置“height: inherit;”能够按预定效果显示，而“height: 100%;”就可能撑开包含框，效果如图 20.8 所示。

```
<style type="text/css">
.box {
    display: inline-block;
    height: 200px;
    width: 45%;
    border: 2px solid #666;
}
.box  div{
    width: 200px;
    background-color: #ccc;
    position: absolute;
}
.height1 { height: 100%;}
.height2 {height: inherit;}
</style>
<div class="box">
    <div class="height1">height: 100%;</div>
</div>
<div class="box">
    <div class="height2">height: inherit;</div>
</div>
```

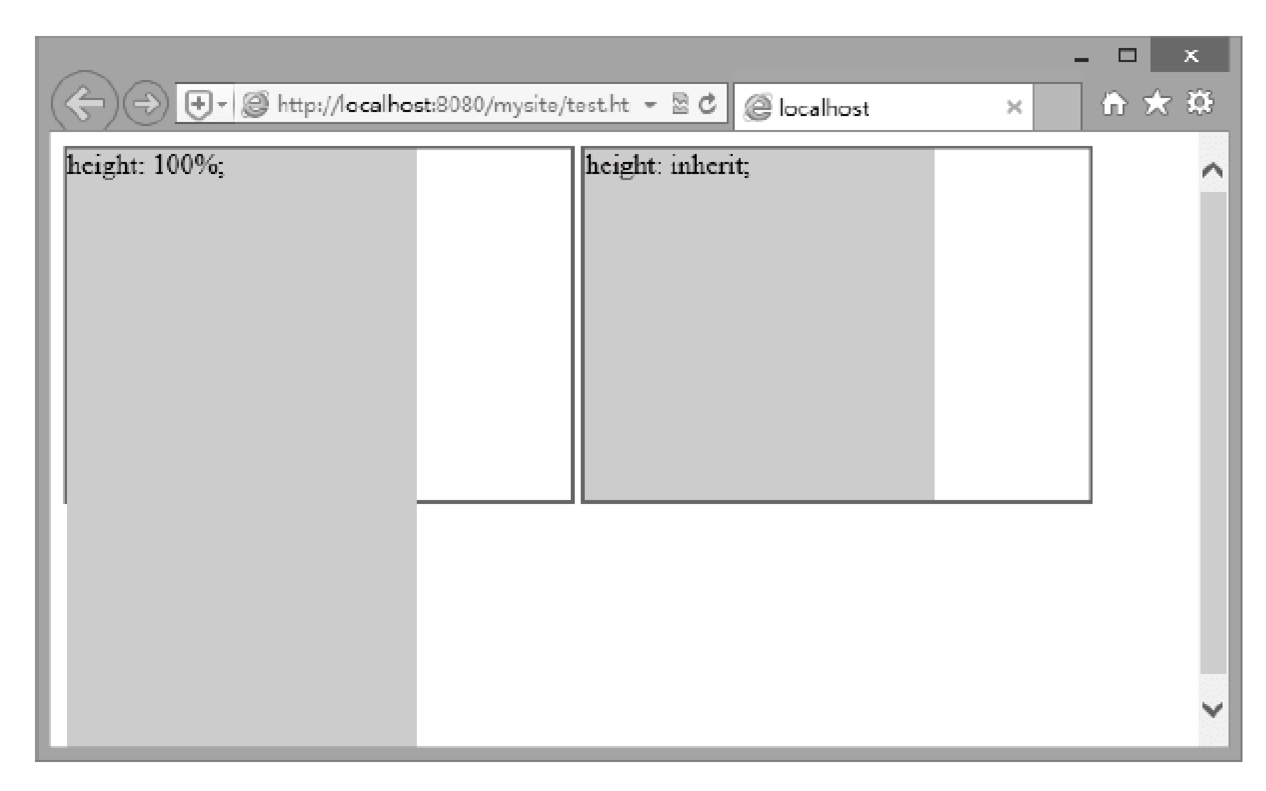


图 20.8　比较 inherit 和 100%高度效果

【补充】

inherit 表示继承属性值，一般用于字体、颜色、背景等；auto 表示自适应，一般用于高度、宽度、外边距和内边距等关于长度的属性。







Note

20.1.7　unset 值

unset 表示擦除用户声明的属性值，所有的属性都可以接受该值。如果属性有继承的值，则该属性的值等同于 inherit，即继承的值不被擦除；如果属性没有继承的值，则该属性的值等同于 initial，即擦除用户声明的值，恢复初始值。

注意：IE 和 Safari 浏览器暂时不支持该属性值。

【示例】下面示例设计四段文本，第一段和第二段位于<div class="box">容器中，设置段落文本显示为 30 像素的蓝色字体，现在擦除第二段和第四段文本样式，则第二段文本显示继承样式，即 12 像素的红色字体，而第四段文本显示初始化样式，即 16 像素的黑色字体，效果如图 20.9 所示。

```
<style type="text/css">
.box {color: red; font-size: 12px;}
p {color: blue; font-size: 30px;}
p.unset {
    color: unset;
    font-size: unset;
}
</style>
<div class="box">
    <p>春眠不觉晓，</p>
    <p class="unset">处处闻啼鸟。</p>
</div>
<p>夜来风雨声，</p>
<p class="unset">花落知多少。</p>
```



图 20.9　比较擦除后的文本效果

20.1.8　all 属性



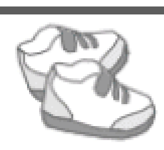
all 属性表示所有 CSS 的属性，但不包括 unicode-bidi 和 direction 这两个 CSS 属性。

注意：IE 浏览器暂时不支持该属性。

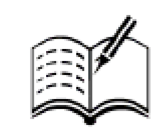
【示例】针对 20.1.7 节示例，我们可以简化 p.unset 类样式。

```
p.unset {
    all: unset;
}
```

如果在样式中，声明的属性非常多，使用 all 会极为方便，避免逐个设置每个属性。




20.2 色彩模式



Note

视频讲解

CSS 2.1 支持 Color Name（颜色名称）、HEX（十六进制颜色值）、RGB，CSS3 新增三种颜色模式：RGBA、HSL 和 HSLA，下面分别进行介绍。

权威参考：http://www.w3.org/TR/css3-color/。

20.2.1 rgba()函数

RGBA 是 RGB 色彩模式的扩展，它在红、绿、蓝三原色通道基础上增加了 Alpha 通道。其语法格式如下所示：

```
rgba(r,g,b,<opacity>)
```

参数说明如下：

☑ r、g、b：分别表示红色、绿色、蓝色三种原色所占的比重。取值为正整数或者百分数。正整数值的取值范围为 0～55，百分数值的取值范围为 0～100.0%。超出范围的数值将被截至其最接近的取值极限。注意，并非所有浏览器都支持使用百分数值。

☑ <opacity>：表示不透明度，取值在 0 到 1 之间。

【示例】下面示例使用 CSS3 的 box-shadow 属性和 rgba()函数为表单控件设置半透明度的阴影，来模拟柔和的润边效果。示例主要代码如下：

```
<style type="text/css">
input, textarea {/*统一文本框样式*/
    padding: 4px;                    /*增加内补白，增大表单对象尺寸，看起来更大方*/
    border: solid 1px #E5E5E5;       /*增加淡淡的边框线*/
    outline: 0;                      /*清除轮廓线*/
    font: normal 13px/100% Verdana, Tahoma, sans-serif;
    width: 200px;                    /*固定宽度*/
    background: #FFFFFF;             /*白色背景*/
    /*设置边框阴影效果*/
    box-shadow: rgba(0, 0, 0, 0.1) 0px 0px 8px;
}
/*定义表单对象获取焦点，鼠标经过时，高亮显示边框*/
input:hover, textarea:hover, input:focus, textarea:focus { border-color: #C9C9C9; }
label {/*定义标签样式*/
    margin-left: 10px;
    color: #999999;
    display:block;                        /*以块状显示，实现分行显示*/
}
.submit input {                           /*定义提交按钮样式*/
    width:auto;                           /*自动调整宽度*/
    padding: 9px 15px;                    /*增大按钮尺寸，看起来更大气*/
    background: #617798;                  /*设计扁平化单色背景*/
    border: 0;                            /*清除边框线*/
    font-size: 14px;                      /*固定字体大小*/
    color: #FFFFFF;                       /*白色字体*/
```

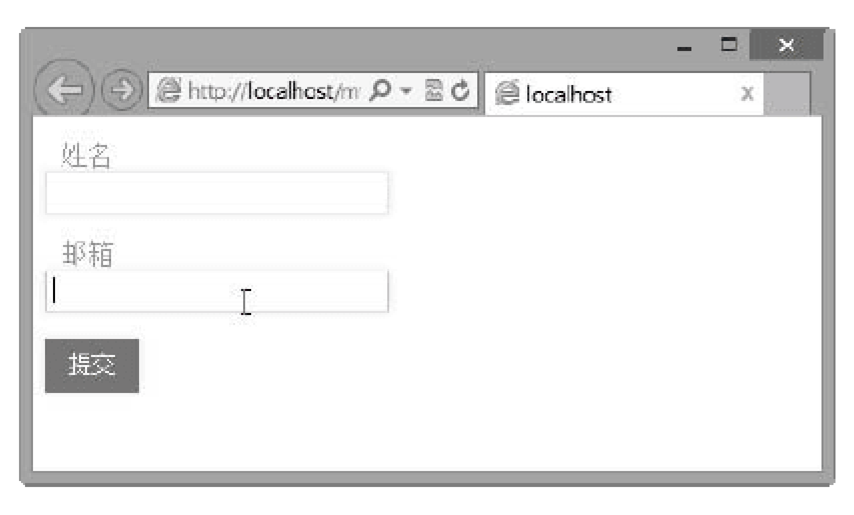


```
    }
    </style>
    <form>
        <p class="name">
            <label for="name">姓名</label>
            <input type="text" name="name" id="name" />
        </p>
        <p class="email">
            <label for="email">邮箱</label>
            <input type="text" name="email" id="email" />
        </p>
        <p class="submit">
            <input type="submit" value="提交" />
        </p>
    </form>
```



Note

预览效果如图 20.10 所示。





示 例 效 果



图 20.10　设计带有阴影边框的表单效果

提示： rgba(0,0,0,0.1)表示不透明度为 0.1 的黑色，这里不宜直接设置为浅灰色，因为对于非白色背景来说，灰色发虚，而半透明效果可以避免这样的情况发生。

20.2.2　hsl()函数



视 频 讲 解

HSL 是一种标准的色彩模式，包括了人类视力所能感知的所有颜色，在屏幕上可以重现 16777216 种颜色，是目前运用最广泛的颜色系统。它通过色调（H）、饱和度（S）和亮度（L）三个颜色通道的叠加来获取各种颜色。其语法格式如下所示。

```
hsl(<length>,<percentage>,<percentage>)
```

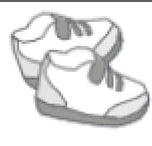
参数说明如下：

☑ <length>表示色调（Hue）。可以为任意数值，用以确定不同的颜色。其中 0（或 360、-360）表示红色，60 表示黄色，120 表示绿色，180 表示青色，240 表示蓝色，300 表示洋红。

☑ <percentage>（第一个）表示饱和度（Saturation），可以为 0～100%。其中 0 表示灰度，即没有使用该颜色；100%表示饱和度最高，即颜色最艳。

☑ <percentage>（第二个）表示亮度（Lightness）。取值为 0～100%。其中 0 最暗，显示为黑色，50%表示均值，100%最亮，显示为白色。

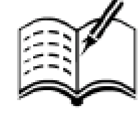
【示例】设计颜色表。先选择一个色值，然后利用调整颜色的饱和度和亮度比重，分别设计不同的配色方案表。在网页设计中，利用这种方法就可以根据网页需要选择恰当的配色方案。使用 HSL 颜色表现方式，可以很轻松地设计网页配色方案表，模拟演示效果如图 20.11 所示。

Note

```
<styletype="text/css">
/*设计表格边框样式，并增加内部间距，以方便观看 */
table{ border: solid 1px red; background:#eee; padding:6px;}
/*设计列标题字体样式*/
th{ color:red; font-size:12px; font-weight:normal;}
/*设计单元格大小尺寸*/
td{ width:80px; height:30px;}
/*第 1 行*/
tr:nth-child(4) td:nth-of-type(1){background:hsl(0,100%,100%);}/*第 1 列*/
tr:nth-child(4) td:nth-of-type(2){background:hsl(0,75%,100%);}/*第 2 列*/
tr:nth-child(4) td:nth-of-type(3){background:hsl(0,50%,100%);}/*第 3 列*/
tr:nth-child(4) td:nth-of-type(4){background:hsl(0,25%,100%);}/*第 4 列*/
tr:nth-child(4) td:nth-of-type(5){background:hsl(0,0%,100%);}/*第 5 列*/
/*第 2 行*/
tr:nth-child(5) td:nth-of-type(1){background:hsl(0,100%,88%);}/*第 1 列*/
tr:nth-child(5) td:nth-of-type(2){background:hsl(0,75%,88%);}/*第 2 列*/
tr:nth-child(5) td:nth-of-type(3){background:hsl(0,50%,88%);}/*第 3 列*/
tr:nth-child(5) td:nth-of-type(4){background:hsl(0,25%,88%);}/*第 4 列*/
tr:nth-child(5) td:nth-of-type(5){background:hsl(0,0%,88%);}/*第 5 列*/
/*第 3 行*/
tr:nth-child(6) td:nth-of-type(1){background:hsl(0,100%,75%);}/*第 1 列*/
tr:nth-child(6) td:nth-of-type(2){background:hsl(0,75%,75%);}/*第 2 列*/
tr:nth-child(6) td:nth-of-type(3){background:hsl(0,50%,75%);}/*第 3 列*/
tr:nth-child(6) td:nth-of-type(4){background:hsl(0,25%,75%);}/*第 4 列*/
tr:nth-child(6) td:nth-of-type(5){background:hsl(0,0%,75%);}/*第 5 列*/
/*第 4 行*/
tr:nth-child(7) td:nth-of-type(1){background:hsl(0,100%,63%);}/*第 1 列*/
tr:nth-child(7) td:nth-of-type(2){background:hsl(0,75%,63%);}/*第 2 列*/
tr:nth-child(7) td:nth-of-type(3){background:hsl(0,50%,63%);}/*第 3 列*/
tr:nth-child(7) td:nth-of-type(4){background:hsl(0,25%,63%);}/*第 4 列*/
tr:nth-child(7) td:nth-of-type(5){background:hsl(0,0%,63%);}/*第 5 列*/
/*第 5 行*/
tr:nth-child(8) td:nth-of-type(1){background:hsl(0,100%,50%);}/*第 1 列*/
tr:nth-child(8) td:nth-of-type(2){background:hsl(0,75%,50%);}/*第 2 列*/
tr:nth-child(8) td:nth-of-type(3){background:hsl(0,50%,50%);}/*第 3 列*/
tr:nth-child(8) td:nth-of-type(4){background:hsl(0,25%,50%);}/*第 4 列*/
tr:nth-child(8) td:nth-of-type(5){background:hsl(0,0%,50%);}/*第 5 列*/
/*第 6 行*/
tr:nth-child(9) td:nth-of-type(1){background:hsl(0,100%,38%);}/*第 1 列*/
tr:nth-child(9) td:nth-of-type(2){background:hsl(0,75%,38%);}/*第 2 列*/
tr:nth-child(9) td:nth-of-type(3){background:hsl(0,50%,38%);}/*第 3 列*/
tr:nth-child(9) td:nth-of-type(4){background:hsl(0,25%,38%);}/*第 4 列*/
tr:nth-child(9) td:nth-of-type(5){background:hsl(0,0%,38%);}/*第 5 列*/
/*第 7 行*/
tr:nth-child(10) td:nth-of-type(1){background:hsl(0,100%,25%);}/*第 1 列*/
tr:nth-child(10) td:nth-of-type(2){background:hsl(0,75%,25%);}/*第 2 列*/
tr:nth-child(10) td:nth-of-type(3){background:hsl(0,50%,25%);}/*第 3 列*/
tr:nth-child(10) td:nth-of-type(4){background:hsl(0,25%,25%);}/*第 4 列*/
tr:nth-child(10) td:nth-of-type(5){background:hsl(0,0%,25%);}/*第 5 列*/
/*第 8 行*/
```





```
tr:nth-child(11) td:nth-of-type(1){background:hsl(0,100%,13%);}/*第 1 列*/
tr:nth-child(11) td:nth-of-type(2){background:hsl(0,75%,13%);}/*第 2 列*/
tr:nth-child(11) td:nth-of-type(3){background:hsl(0,50%,13%);}/*第 3 列*/
tr:nth-child(11) td:nth-of-type(4){background:hsl(0,25%,13%);}/*第 4 列*/
tr:nth-child(11) td:nth-of-type(5){background:hsl(0,0%,13%);}/*第 5 列*/
/*第 9 行*/
tr:nth-child(12) td:nth-of-type(1){background:hsl(0,100%,0%);}/*第 1 列*/
tr:nth-child(12) td:nth-of-type(2){background:hsl(0,75%,0%);}/*第 2 列*/
tr:nth-child(12) td:nth-of-type(3){background:hsl(0,50%,0%);}/*第 3 列*/
tr:nth-child(12) td:nth-of-type(4){background:hsl(0,25%,0%);}/*第 4 列*/
tr:nth-child(12) td:nth-of-type(5){background:hsl(0,0%,0%);}/*第 5 列*/
</style>
<table class="hslexample">
    <tbody>
        <tr>
            <th> </th><th colspan="5">色相：H=0 Red </th>
        </tr>
        <tr>
            <th> </th><th colspan="5">饱和度 (&rarr;)</th>
        </tr>
        <tr>
            <th>亮度 (&darr;)</th>
            <th>100% </th><th>75% </th><th>50% </th><th>25% </th><th>0% </th>
        </tr>
        ……
    </tbody>
</table>
```

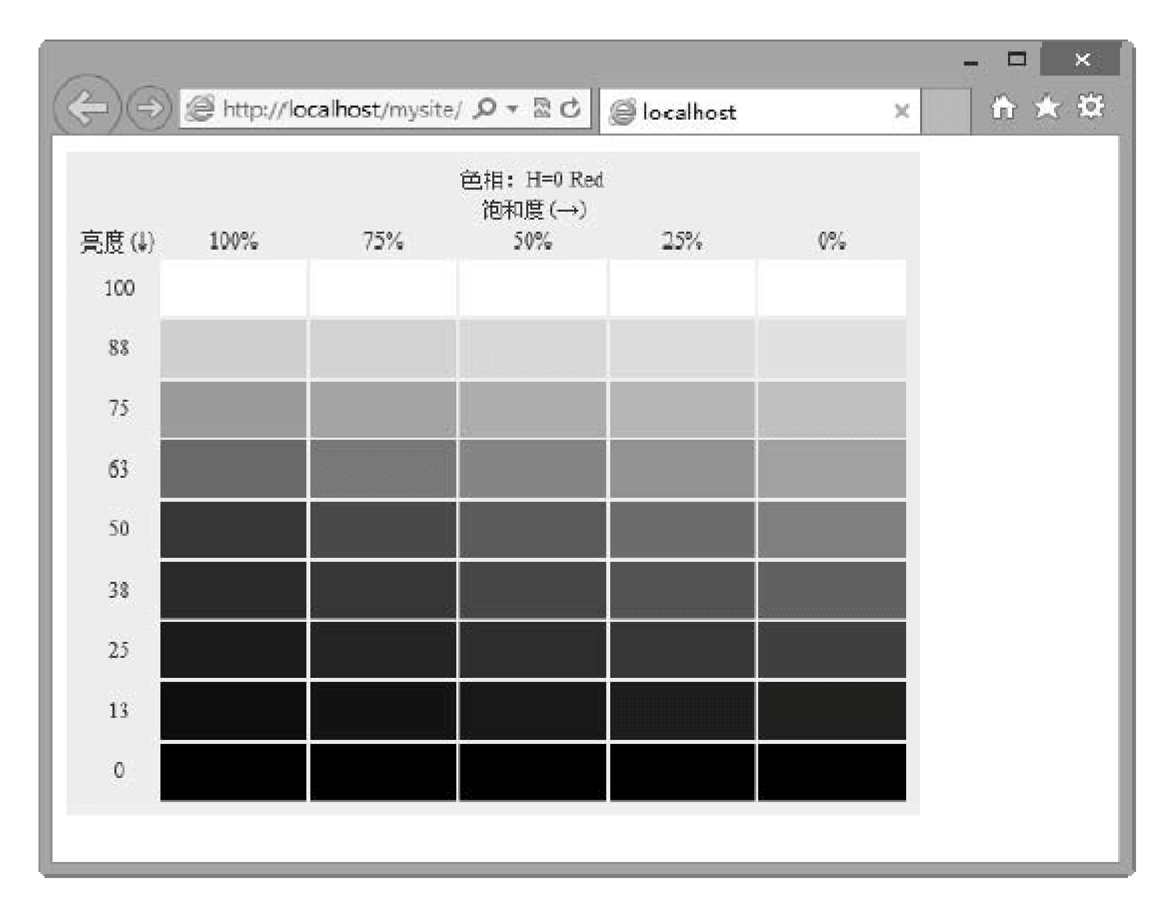


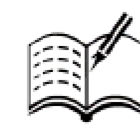
图 20.11　使用 HSL 颜色值设计颜色表



在上面代码中，“tr:nth-child(4) td:nth-of-type(1)”中的 tr:nth-child(4)子选择器表示选择行，而 td:nth-of-type(1)表示选择单元格（列）。其他行选择器结构依此类推。在“background:hsl(0,0%,0%);”声明中，hsl()函数的第一个参数值 0 表示色相值，第二个参数值 0%表示饱和度，第三个参数值 0%表示亮度。



视 频 讲 解



Note

20.2.3 hsla()函数

HSLA 是 HSL 色彩模式的扩展，在色相、饱和度、亮度三要素基础上增加了不透明度参数。使用 HSLA 色彩模式，可以定义不同透明效果。其语法格式如下。

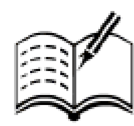
```
hsla(<length>,<percentage>,<percentage>,<opacity>)
```

其中前三个参数与 hsl()函数参数含义和用法相同，第四个参数<opacity>表示不透明度，取值在 0 到 1 之间。

【示例】下面示例设计一个简单的登录表单，表单对象的边框色使用#fff 值进行设置，定义为白色；表单对象的阴影色使用 rgba(0,0,0,0.1)值进行设置，定义为非常透明的黑色；字体颜色使用 hsla(0,0%,100%,0.9)值进行设置，定义为轻微透明的白色。预览效果如图 20.12 所示。

```
<style  type="text/css">
body{ /* 为页面添加背景图像，显示在中央顶部位置，并列完全覆盖窗口 */
    background: #eedfcc url(images/bg.jpg) no-repeat center top;
    background-size: cover;
}
.form { /* 定义表单框的样式 */
    width: 300px;                                    /* 固定表单框的宽度 */
    margin: 30px auto;                               /* 居中显示 */
    border-radius: 5px;                              /* 设计圆角效果 */
    box-shadow: 0 0 5px rgba(0,0,0,0.1),             /* 设计润边效果 */
                0 3px 2px rgba(0,0,0,0.1);           /* 设计淡淡的阴影效果 */
}
.form p { /* 定义表单对象外框圆角、白边显示 */
    width: 100%;
    float: left;
    border-radius: 5px;
    border: 1px solid #fff;
}
/* 定义表单对象样式 */
.form input[type=text],
.form input[type=password] {
    /* 固定宽度和大小 */
    width: 100%;
    height: 50px;
    padding: 0;
    /*增加修饰样式 */
    border: none;                                    /* 移除默认的边框样式*/
    background: rgba(255,255,255,0.2);               /* 增加半透明的白色背景 */
    box-shadow: inset 0 0 10px rgba(255,255,255,0.5); /* 为表单对象设计高亮效果 */
    /* 定义字体样式*/
    text-indent: 10px;
    font-size: 16px;
    color:hsla(0,0%,100%,0.9);
    text-shadow: 0 -1px 1px rgba(0,0,0,0.4);         /* 为文本添加阴影，设计立体效果 */
}
.form input[type=text] {                 /* 设计用户名文本框底部边框样式，并设计顶部圆角 */
```





```
        border-bottom: 1px solid rgba(255,255,255,0.7);
        border-radius: 5px 5px 0 0;
    }
    .form input[type=password] {        /* 设计密码域文本框顶部边框样式，并设计底部圆角 */
        border-top: 1px solid rgba(0,0,0,0.1);
        border-radius: 0 0 5px 5px;
    }
    /* 定义表单对象被激活，或者鼠标经过时，增亮背景色，并清除轮廓线 */
    .form input[type=text]:hover,
    .form input[type=password]:hover,
    .form input[type=text]:focus,
    .form input[type=password]:focus {
        background: rgba(255,255,255,0.4);
        outline: none;
    }
    </style>
    <form class="form">
        <p>
            <input type="text" id="login" name="login" placeholder="用户名">
            <input type="password" name="password" id="password" placeholder="密码">
        </p>
    </form>
```

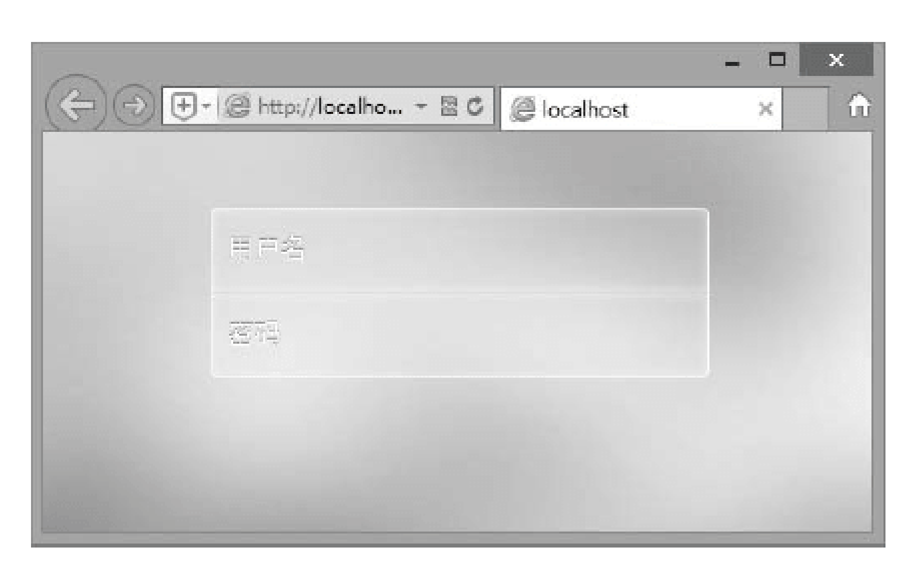


图 20.12　设计登录表单



示 例 效 果

20.2.4　opacity 属性



视 频 讲 解

opacity 属性定义元素对象的不透明度。其语法格式如下所示：

```
opacity: <alphavalue> | inherit;
```

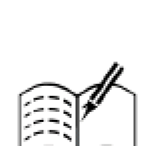
取值简单说明如下：

☑ <alphavalue>为由浮点数字和单位标识符组成的长度值。不可为负值，默认值为 1。opacity 取值为 1 时，则元素是完全不透明的；取值为 0 时，元素是完全透明的，不可见的；介于 1 到 0 之间的任何值都表示该元素的不透明度。如果超过了这个范围，其计算结果将截取到与之最相近的值。

☑ inherit 表示继承父辈元素的不透明性。

【示例】 下面示例设计<div class="bg">对象铺满整个窗口，显示为黑色背景，不透明度为 0.7，这样可以模拟一种半透明的遮罩效果；再使用 CSS 定位属性设计<div class="login">对象显示在上面。示例主要代码如下：





Note

```
<style type="text/css">
body {margin: 0; padding: 0;}
div { position: absolute; }
.bg {
    width: 100%;
    height: 100%;
    background: #000;
    opacity: 0.7;
    filter: alpha(opacity=70);}
. login {
    text-align:center;
    width:100%;
    top: 20%;
}
</style>
<div class="web"><img src="images/bg.png" /></div>
<div class="bg"></div>
<div class="login"><img src="images/login.png"  /></div>
```

演示效果如图 20.13 所示。

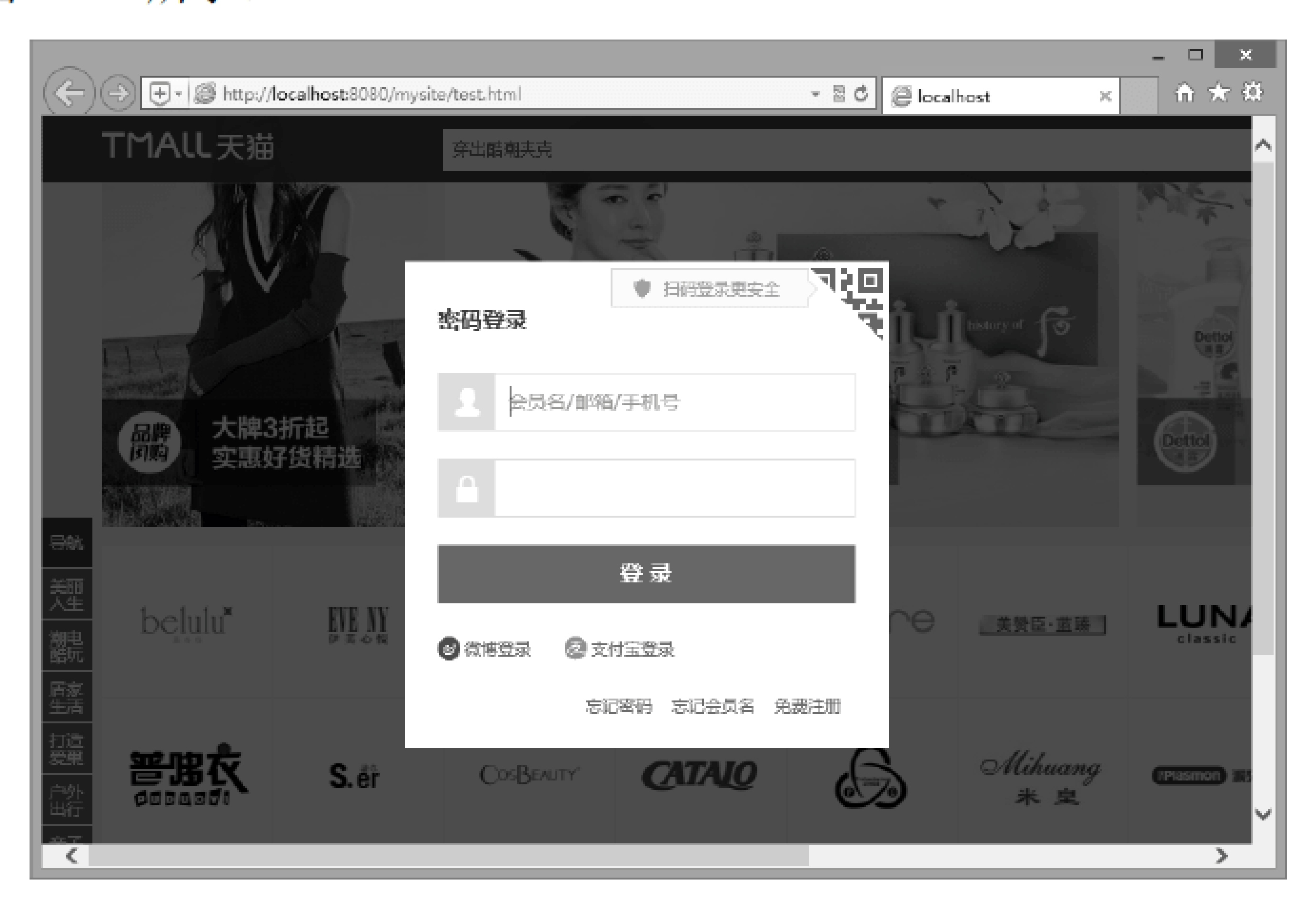


图 20.13　设计半透明的背景布效果

注意：使用色彩模式函数的 alpha 通道可以针对元素的背景色或文字颜色单独定义不透明度，而 opacity 属性只能为整个对象定义不透明度。

20.2.5　transparent 值

视 频 讲 解

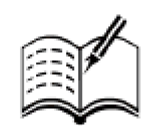
transparent 属性值用来指定全透明色彩，等效于 rgba(0,0,0,0)值。

【示例】下面示例使用 CSS 的 border 设计三角形效果，通过 transparent 颜色值让部分边框透明显示，代码如下所示：

```
<style type="text/css">
#demo {
    width: 0; height: 0;
```



```
        border-left: 50px solid transparent;
        border-right: 50px solid transparent;
        border-bottom: 100px solid red;
    }
    </style>
    <div id="demo"></div>
```



Note

效果如图 20.14 所示。

通过调整各边颜色设置，或者调整各边宽度，可以设计不同角度的三角形，或者设计直角等不同形状。

☑　设计向右三角形

```
    #demo {
        width: 0; height: 0;
        border-top: 50px solid transparent;
        border-left: 100px solid red;
        border-bottom: 50px solid transparent;
    }
```

☑　设计直角三角形

```
    #demo {
        width: 0; height: 0;
        border-top: 100px solid red;
        border-right: 100px solid transparent;
    }
```

☑　设计梯形

效果如图 20.15 所示。

```
    #demo {
        height: 0;
        width: 120px;
        border-bottom: 120px solid #ec3504;
        border-left: 60px solid transparent;
        border-right: 60px solid transparent;
    }
```

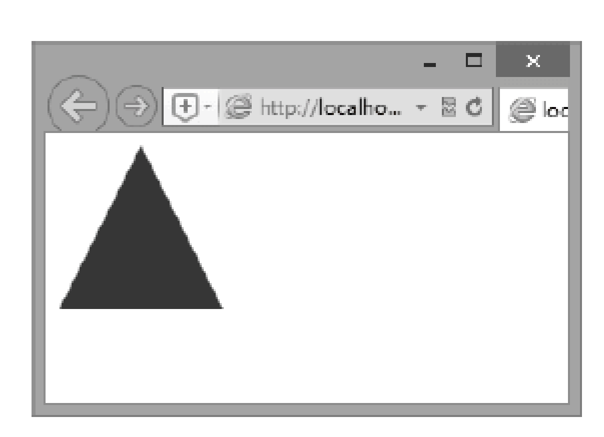


图 20.14　设计三角形效

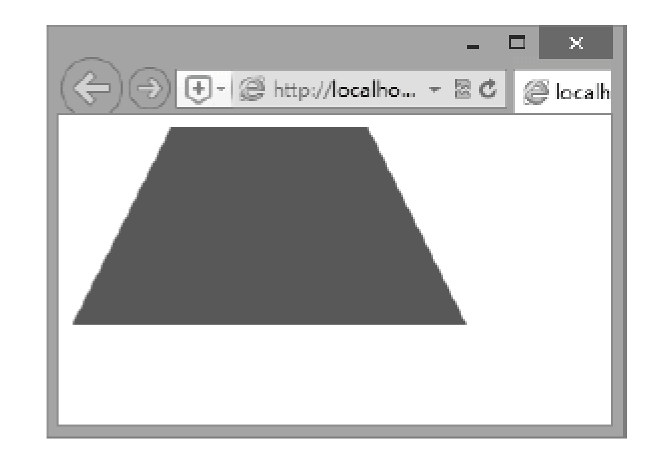


图 20.15　设计梯形效果

20.2.6　currentColor 值

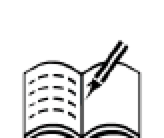
在 CSS 中，border-color、box-shadow 和 text-decoration-color 属性的默认值是 color 属性的值。



视 频 讲 解

【示例 1】下面示例中为段落文本增加边框线，边框线的颜色为“color:red;”，显示为红色。





Note

```
<style type="text/css">
p {
    border:solid 2px;
    color:red;
}
</style>
<p>春眠不觉晓，处处闻啼鸟。夜来风雨声，花落知多少。</p>
```

在 CSS 1 和 CSS 2 中，却没有为此定义一个相应的关键字。为此 CSS3 扩展了颜色值，包含 currentColor 关键字，并用于所有接受颜色的属性上。currentColor 表示 color 属性的值。

【示例 2】在下面示例中，设计图标背景颜色值为 currentColor，这样在网页中随着链接文本的字体颜色不断变化，图标的颜色也跟随链接文本的颜色变化而变化，确保整体导航条色彩一致性，达到图文合一的境界，效果如图 20.16 所示。

```
<style type="text/css">
.icon {
    display: inline-block;
    width: 16px; height: 20px;
    background-image: url(images/sprite_icons.png);
    background-color: currentColor; /* 使用当前颜色控制图标的颜色 */
}
.icon1 { background-position: 0 0; }
.icon2 { background-position: -20px 0; }
.icon3 { background-position: -40px 0; }
.icon4 { background-position: -60px 0; }
.link { margin-right: 15px; }
.link:hover { color: red; }/* 虽然改变的是文字颜色，但是图标颜色也一起变化了 */
</style>
<a href="##" class="link"><i class="icon icon1"></i>首页</a>
<a href="##" class="link"><i class="icon icon2"></i>刷新</a>
<a href="##" class="link"><i class="icon icon3"></i>收藏</a>
<a href="##" class="link"><i class="icon icon4"></i>展开</a>
```



示例效果



图 20.16　设计图标背景色为 currentColor

提示：如果为 color 属性设置为 currentColor，则相当于 color: inherit。

20.3　文本阴影

CSS3 使用 text-shadow 属性可以给文本添加阴影效果，到目前为止，Safari、Firefox、Chrome 和 Opera 等主流浏览器都支持该功能。







20.3.1 定义 text-shadow

text-shadow 属性是在 CSS2 中定义的，在 CSS 2.1 中被删除，在 CSS3 的 Text 模块中又恢复。基本语法如下所示：

```
text-shadow:none | <length>{2,3} && <color>?
```

取值简单说明如下：

- ☑ none：无阴影，为默认值。
- ☑ <length>①：第 1 个长度值用来设置对象的阴影水平偏移值。可以为负值。
- ☑ <length>②：第 2 个长度值用来设置对象的阴影垂直偏移值。可以为负值。
- ☑ <length>③：如果提供了第 3 个长度值，则用来设置对象的阴影模糊值。不允许负值。
- ☑ <color>：设置对象的阴影的颜色。

【示例】下面为段落文本定义一个简单的阴影效果，演示效果如图 20.17 所示。

```
<style type="text/css">
p {
    text-align: center;
    font: bold 60px helvetica, arial, sans-serif;
    color: #999;
    text-shadow: 0.1em 0.1em #333;
}
</style>
<p>HTML5+CSS3</p>
```



图 20.17 定义文本阴影

“text-shadow: 0.1em 0.1em #333;”声明了右下角文本阴影效果，如果把投影设置到右上角，则可以这样声明，效果如图 20.18 所示。

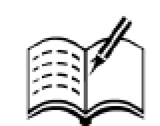


图 20.18 定义左上角阴影

```
p {text-shadow: -0.1em -0.1em #333;}
```

同理，如果设置阴影在文本的左下角，则可以设置如下样式，演示效果如图 20.19 所示。





Note

```
p {text-shadow: -0.1em 0.1em #333;}
```



图 20.19　定义左下角阴影

也可以增加模糊效果的阴影，效果如图 20.20 所示。

```
p{ text-shadow: 0.1em 0.1em 0.3em #333; }
```

或者定义如下模糊阴影效果，效果如图 20.21 所示。

```
p{ text-shadow: 0.1em 0.1em 0.2em black; }
```





示例效果



图 20.20　定义模糊阴影



图 20.21　定义模糊阴影



提示：在 text-shadow 属性的第一个值和第二个值中，正值偏右或偏下，负值偏左或偏上。在阴影偏移之后，可以指定一个模糊半径。模糊半径是个长度值，指出模糊效果的范围。如何计算模糊效果的具体算法并没有指定。在阴影效果的长度值之前或之后还可以选择指定一个颜色值。颜色值会被用作阴影效果的基础。如果没有指定颜色，那么将使用 color 属性值来替代。



视频讲解

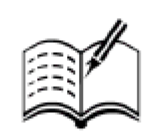
20.3.2　案例：设计特效字

下面结合示例介绍如何灵活使用 text-shadow 属性设计特效文字效果。

【示例 1】下面示例通过阴影把文本颜色与背景色区分开来，让字体看起来更清晰，代码如下：

```
<style type="text/css">
p {
    text-align: center;
    font: bold 60px helvetica, arial, sans-serif;
    color: #fff;
    text-shadow: black 0.1em 0.1em 0.2em;
}
</style>
<p>HTML5+CSS3</p>
```





演示效果如图 20.22 所示。

【示例 2】下面示例演示了如何为红色文本定义三个不同颜色的阴影，演示效果如图 20.23 所示。当使用 text-shadow 属性定义多色阴影时，每个阴影效果必须指定阴影偏移，而模糊半径、阴影颜色是可选参数。

```
<style type="text/css">
p {
    text-align: center;
    font:bold 60px helvetica, arial, sans-serif;
    color: red;
    text-shadow: 0.2em 0.5em 0.1em #600,
        -0.3em 0.1em 0.1em #060,
        0.4em -0.3em 0.1em #006;
}
</style>
<p>HTML5+CSS3</p>
```



图 20.22　使用阴影增加前景色和背景色对比度



图 20.23　定义多色阴影

提示：text-shadow 属性可以接受以逗号分隔的阴影效果列表，并应用到该元素的文本上。阴影效果按照给定的顺序应用，因此可能出现互相覆盖，但是它们永远不会覆盖文本本身。阴影效果不会改变框的尺寸，但可能延伸到它的边界之外。阴影效果的堆叠层次和元素本身的层次是一样的。

【示例 3】下面演示把阴影设置到文本线框的外面，代码如下：

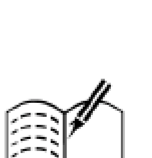
```
<style type="text/css">
p {
    text-align: center;
    font:bold 60px helvetica, arial, sans-serif;
    color: red;
    border:solid 1px red;
    text-shadow: 0.5em 0.5em 0.1em #600,
        -1em 1em 0.1em #060,
        0.8em -0.8em 0.1em #006;
}
</style>
<p>HTML5+CSS3</p>
```

演示效果如图 20.24 所示。

【示例 4】借助阴影效果列表机制，可以使用阴影叠加出燃烧的文字特效，代码如下：

```
<style type="text/css">
body {background:#000;}
p {
```





Note

```
        text-align: center;
        font:bold 60px helvetica, arial, sans-serif;
        color: red;
        text-shadow: 0 0 4px white,
            0 -5px 4px #ff3,
            2px -10px 6px #fd3,
            -2px -15px 11px #f80,
            2px -25px 18px #f20;
    }
    </style>
    <p>HTML5+CSS3</p>
```

演示效果如图 20.25 所示。




图 20.24　定义多色阴影



图 20.25　定义燃烧的文字特效

【示例 5】text-shadow 属性可以使用在“:first-letter”和“:first-line”伪元素上。同时还可以利用该属性设计立体文本。使用阴影叠加出的立体文本特效代码如下：

```
<style type="text/css">
body { background: #000; }
p {
    text-align: center;
    padding: 24px;
    margin: 0;
    font-family: helvetica, arial, sans-serif;
    font-size: 80px;
    font-weight: bold;
    color: #D1D1D1;
    background: #CCC;
    text-shadow: -1px -1px white,
        1px 1px #333;
}
</style>
<p>HTML5+CSS3</p>
```

演示效果如图 20.26 所示。通过左上和右下各添加一个 1 像素错位的补色阴影，营造一种淡淡的立体效果。

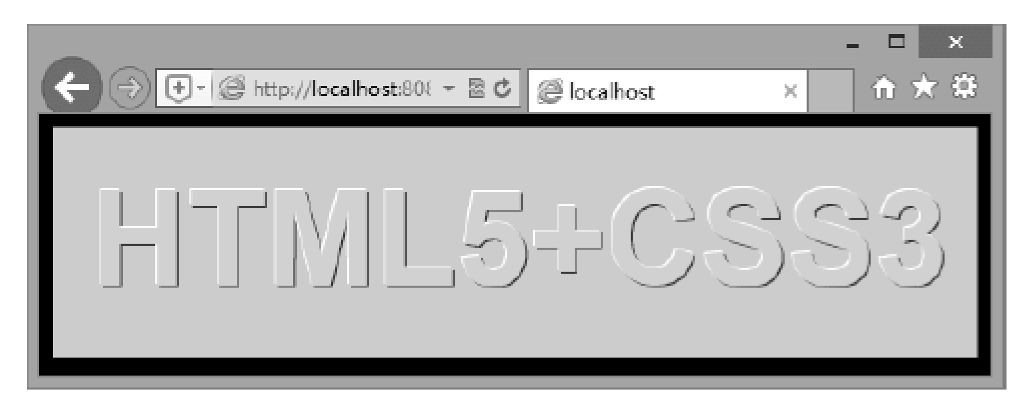
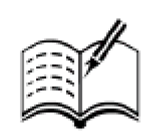


图 20.26　定义凸起的文字效果





【示例 6】 反向思维，利用上面示例的设计思路，也可以设计一种凹体效果，设计方法就是把上面示例中左上和右下阴影颜色颠倒，主要代码如下：

```
<style type="text/css">
body { background: #000; }
p {
    text-align: center;
    padding: 24px;
    margin: 0;
    font-family: helvetica, arial, sans-serif;
    font-size: 80px;
    font-weight: bold;
    color: #D1D1D1;
    background: #CCC;
    text-shadow: 1px 1px white,
        -1px -1px #333;
}
</style>
<p>HTML5+CSS3</p>
```

演示效果如图 20.27 所示。

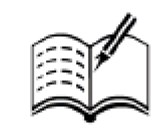


图 20.27　定义凹下的文字效果

【示例 7】 使用 text-shadow 属性还可以为文本描边，设计方法是分别为文本四个边添加 1 像素的实体阴影，代码如下所示：

```
<style type="text/css">
body { background: #000; }
p {
    text-align: center;
    padding:24px;
    margin:0;
    font-family: helvetica, arial, sans-serif;
    font-size: 80px;
    font-weight: bold;
    color: #D1D1D1;
    background:#CCC;
    text-shadow: -1px 0 black,
        0 1px black,
        1px 0 black,
        0 -1px black;
}
</style>
<p>HTML5+CSS3</p>
```





Note

演示效果如图 20.28 所示。

【示例 8】设计阴影不发生位移，同时定义阴影模糊显示，这样就可以模拟出文字外发光效果，代码如下：

```
<style type="text/css">
body { background: #000; }
p {
    text-align: center;
    padding:24px;
    margin:0;
    font-family: helvetica, arial, sans-serif;
    font-size: 80px;
    font-weight: bold;
    color: #D1D1D1;
    background:#CCC;
    text-shadow: 0 0 0.2em #F87,
        0 0 0.2em #F87;
}
</style>

<p>HTML5+CSS3</p>
```

演示效果如图 20.29 所示。

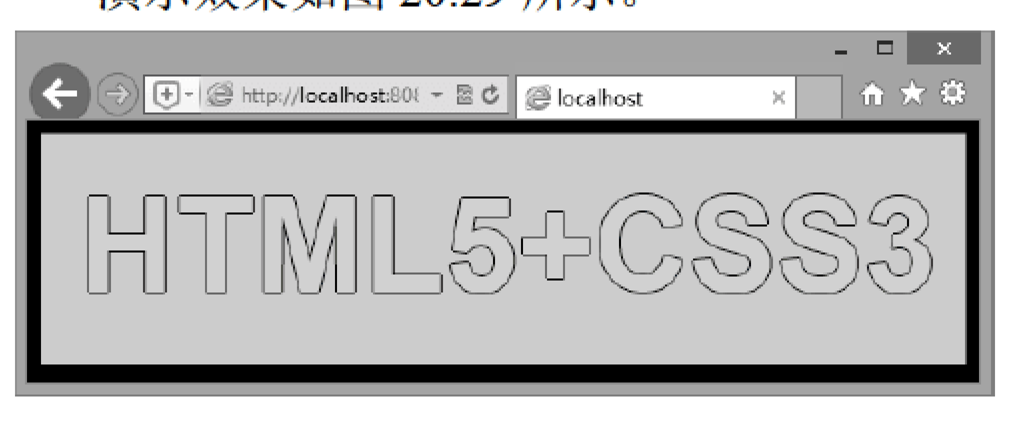


图 20.28　定义描边文字效果



图 20.29　定义外发光文字效果

示例效果

20.4　内容生成和替换



权威参考

content 属性属于内容生成和替换模块，可以为匹配的元素动态生成内容。这样就能够满足在 CSS 样式设计中临时添加非结构性的样式服务标签，或者添加补充说明性内容等。

权威参考：http://www.w3.org/TR/css3-content/。



视频讲解

20.4.1　定义 content

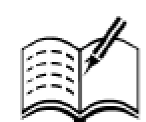
content 属性的简明语法如下所示：

```
content: normal | string | attr() | url() | counter() | none;
```

取值简单说明如下：

☑　normal：默认值。表现与 none 值相同。





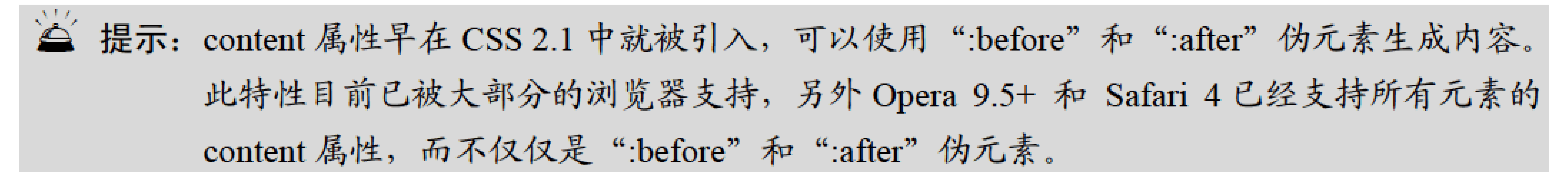
☑　string：插入文本内容。

☑　attr()：插入元素的属性值。

☑　url()：插入一个外部资源，如图像、音频、视频或浏览器支持的其他任何资源。

☑　counter()：计数器，用于插入排序标识。

☑　none：无任何内容。



提示： content 属性早在 CSS 2.1 中就被引入，可以使用“:before”和“:after”伪元素生成内容。此特性目前已被大部分的浏览器支持，另外 Opera 9.5+ 和 Safari 4 已经支持所有元素的 content 属性，而不仅仅是“:before”和“:after”伪元素。

在 CSS3 Generated Content 工作草案中，content 属性添加了更多的特征，例如插入以及移除文档内容的能力，可以创建脚注、段落注释等。但目前还没有浏览器支持 content 的扩展功能。

【示例 1】 下面示例使用 content 属性为页面对象添加外部图像，演示效果如图 20.30 所示。

```
<style type="text/css">
div:after {
    border: solid 10px red;
    content: url(images/bg.png); /*在 div 元素内添加图片*/
}
</style>
<div>
 <h2>动态生成的图片</h2>
</div>
```

注意： content 属性通常与“:after”及“:before”伪元素一起使用，在对象前或后显示内容。

【示例 2】 下面示例使用 content 属性把超链接的 URL 字符串动态显示在页面中，演示效果如图 20.31 所示。

```
<style type="text/css">
a:after {
   content: attr(href);
}
</style>
<a   href="http://www.baidu.com/">百度</a>
```



图 20.30　动态生成图像演示效果

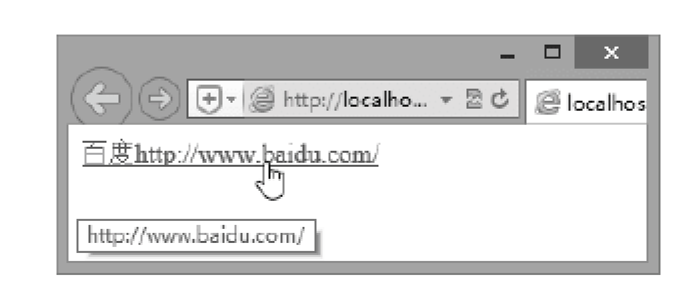
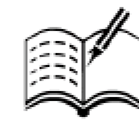


图 20.31　把属性值显示在页面中



视频讲解



Note

20.4.2 案例：应用 content

下面结合多个示例练习 content 在网页中的应用。

【示例 1】下面示例使用 content 属性，配合 CSS 计数器设计多层嵌套有序列表序号设计，效果如图 20.32 所示。

```
<style type="text/css">
ol { list-style:none;}                          /*清除默认的序号*/
li:before {color:#f00; font-family:Times New Roman;} /*设计层级目录序号的字体样式*/
li{counter-increment:a 1;}                      /*设计递增函数 a，递增起始值为 1 */
li:before{content:counter(a)". ";}              /*把递增值添加到列表项前面*/
li li{counter-increment:b 1;}                   /*设计递增函数 b，递增起始值为 1 */
li li:before{content:counter(a)"."counter(b)". ";}  /*把递增值添加到二级列表项前面*/
li li li{counter-increment:c 1;}                /*设计递增函数 c，递增起始值为 1 */
li li li:before{content:counter(a)"."counter(b)"."counter(c)". ";}   /*把递增值添加到三级列表项前面*/
</style>
<h1>网站导航</h1>
<ol>
    <li>新闻
        <ol>
            <li>国际新闻</li>
            <li>国内新闻
                <ol>
                    <li>互联网/科技</li>
                    <li>财经/理财</li>
                </ol>
            </li>
        </ol>
    </li>
    <li>交互</li>
</ol>
```

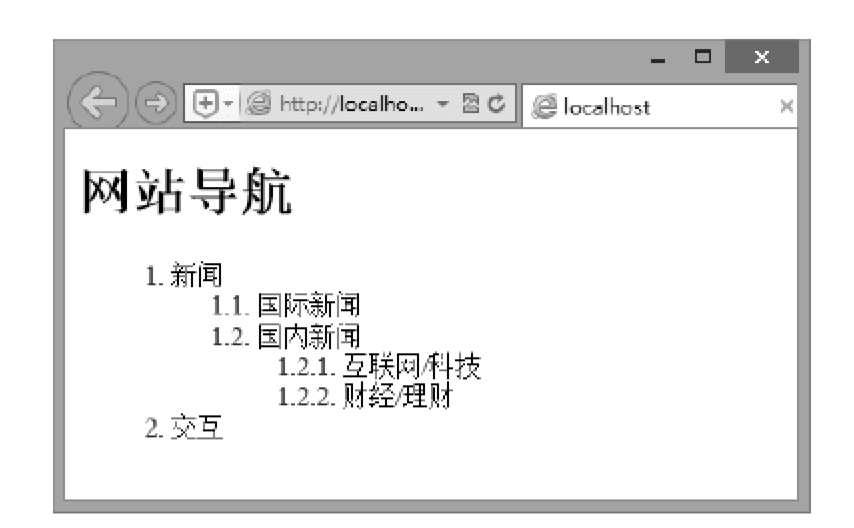


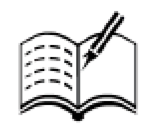
图 20.32　使用 CSS 技巧设计多级层级目录序号

【示例 2】下面示例使用 content 为引文动态添加引号，演示效果如图 20.33 所示。

```
<style type="text/css">
/*  为不同语言指定引号的表现  */
:lang(en) > q {quotes:'"' '"';}
:lang(no) > q {quotes:"«" "»";}
:lang(ch) > q {quotes:"“" "”";}
/*  在 q 标签的前后插入引号  */
```



```
q:before {content:open-quote;}
q:after  {content:close-quote;}
</style>
<p lang="no"><q>HTML5+CSS3 从入门到精通</q></p>
<p lang="en"><q>CSS Generated Content Module Level 3</p>
<p lang="ch"><q>CSS 生成内容模块 3.0</q></p>
```



Note

【示例 3】下面示例使用 content 为超链接动态添加类型图标，演示效果如图 20.34 所示。

```
<style type="text/css">
a[href $=".pdf"]:after {
    content:url(images/icon_pdf.png);
}
a[rel = "external"]:after {
    content:url(images/icon_link.png);
}
</style>
<a href="http://www.book.com/1688.pdf">《HTML5+CSS3 从入门到精通》</a><br>
<a href="http://www.book.com/1688/" rel="external">《HTML5+CSS3 从入门到精通》</a>
```



图 20.33　动态生成引号



图 20.34　动态生成超链接类型图标



示 例 效 果

20.5　网 络 字 体

CSS3 允许用户通过@font-face 规则加载网络字体文件，实现自定义字体类型的功能。@font-face 规则在 CSS3 规范中属于字体模块。

权威参考：http://www.w3org/TR/css3-fonts/#font-face。

20.5.1　使用@font-face



视 频 讲 解

@font-face 规则的语法格式如下：

```
@font-face { <font-description> }
```

@font-face 规则的选择符是固定的，用来引用网络字体文件。<font-description>是一个属性名值对，格式类似如下样式：

```
descriptor: value;
descriptor: value;
descriptor: value;
descriptor: value;
[...]
descriptor: value;
```

属性及其取值说明如下。

☑ font-family：设置文本的字体名称。
☑ font-style：设置文本样式。
☑ font-variant：设置文本是否大小写。
☑ font-weight：设置文本的粗细。
☑ font-stretch：设置文本是否横向拉伸变形。
☑ font-size：设置文本字体大小。
☑ src：设置自定义字体的相对或者绝对路径。注意，该属性只用在@font-face 规则里。

提示：事实上，IE 5 已经开始支持该属性，但是只支持微软自有的.eot（Embedded Open Type）字体格式，而其他浏览器直到现在都不支持这一字体格式。不过，从 Safari 3.1 开始，用户可以设置.ttf（TrueType）和.otf（OpenType）两种字体作为自定义字体了。考虑到浏览器的兼容性，在使用时建议同时定义.eot 和.ttf，以便能够兼容所有主流浏览器。

【示例】下面是一个简单的示例，演示如何使用@font-face 规则在页面中使用网络字体。示例代码如下：

```
<style type="text/css">
/* 引入外部字体文件 */
@font-face {
    /* 选择默认的字体类型 */
    font-family: "lexograph";
    /* 兼容 IE */
    src: url(http://randsco.com//fonts/lexograph.eot);
    /* 兼容非 IE */
    src: local("Lexographer"), url(http://randsco.com/fonts/lexograph.ttf) format("truetype");
}
h1 {
    /* 设置引入字体文件中的 lexograph 字体类型 */
    font-family: lexograph, verdana, sans-serif;
    font-size:4em;}
</style>
<h1>http://www.baidu.com/</h1>
```

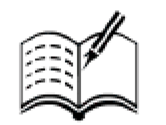
演示效果如图 20.35 所示。



图 20.35 设置为 lexograph 字体类型的文字

提示：嵌入外部字体需要考虑用户带宽问题，因为一个中文字体文件少的有几个 MB，大的有十几个 MB，这么大的字体文件下载过程会出现延迟，同时服务器也不能忍受如此频繁的申请下载。如果只是想标题使用特殊字体，最好设计成图片。





20.5.2　案例：设计字体图标

本节示例通过@font-face 规则引入外部字体文件 glyphicons-halflings-regular.eot，然后定义几个字体图标，嵌入在导航菜单项目中，效果如图 20.36 所示。



图 20.36　设计包含字体图标的导航菜单



示例主要代码如下所示：

```
<style type="text/css">
/* 引入外部字体文件 */
@font-face {
    font-family: 'Glyphicons Halflings';      /* 选择默认的字体类型 */
    /* 外部字体文件列表 */
    src: url('fonts/glyphicons-halflings-regular.eot');
    src: url('fonts/glyphicons-halflings-regular.eot?#iefix') format('embedded-opentype'),
        url('fonts/glyphicons-halflings-regular.woff2') format('woff2'),
        url('fonts/glyphicons-halflings-regular.woff') format('woff'),
        url('fonts/glyphicons-halflings-regular.ttf') format('truetype'),
        url('fonts/glyphicons-halflings-regular.svg#glyphicons_halflingsregular') format('svg');
}
/* 定义字体图标样式 */
.glyphicon {
    position: relative;                       /* 相对定位 */
    top: 1px;                                 /* 相对向上偏移 1 个像素 */
    display: inline-block;                    /* 行内块显示 */
    font-family: 'Glyphicons Halflings';      /* 定义字体类型 */
    font-style: normal;                       /* 字体样式 */
    font-weight: normal;                      /* 字体粗细 */
    line-height: 1;                           /* 定义行高，清除文本行对图标的影响 */
    -webkit-font-smoothing: antialiased;      /* 兼容谷歌浏览器解析 */
    -moz-osx-font-smoothing: grayscale;       /* 兼容 Firefox 浏览器解析 */
}
.glyphicon-home:before { content: "\e021"; }
.glyphicon-user:before { content: "\e008"; }
.glyphicon-search:before { content: "\e003"; }
.glyphicon-plus:before { content: "\e081"; }
span {/* 定义字体图标标签样式 */
    font-size: 16px;
    color: red;
}
ul {/* 定义导航列表框样式，清除默认样式 */
    margin: 0;
    padding: 0;
    list-style: none;
```




Note

```
}
li {/* 定义列表项目样式，水平并列显示 */
    float: left;
    padding: 6px 12px;
    margin: 3px;
    border: solid 1px hsla(359,93%,69%,0.6);
    border-radius: 6px;
}
li a {/* 定义超链接文本样式 */
    font-size: 16px;
    color: red;
    text-decoration: none;
}
</style>
<ul>
    <li><span class="glyphicon glyphicon-home"></span> <a href="#">主页</a></li>
    <li><span class="glyphicon glyphicon-user"></span> <a href="#">登录</a></li>
    <li><span class="glyphicon glyphicon-search"></span> <a href="#">搜索</a></li>
    <li><span class="glyphicon glyphicon-plus"></span> <a href="#">添加</a></li>
</ul>
```

20.6 案例实战

本节将以案例形式实战练习 CSS3 新增的文本属性。

视频讲解

20.6.1 设计黑科技网站首页

本示例将模拟一个黑科技网站的首页，借助 text-shadow 属性设计阴影效果，通过颜色的搭配，营造一幅静谧而又神秘的画面，使用两幅 PNG 图像对页面效果进行装饰和点缀，最后演示效果如图 20.37 所示。





线上阅读

图 20.37 设计黑科技网站首页

具体代码解析请扫码学习。



视频讲解

Note

20.6.2　设计消息提示框

本节将借助 CSS3 增强的文本特性以及相关动画功能，设计一个纯 CSS 的消息提示框，效果如图 20.38 所示。

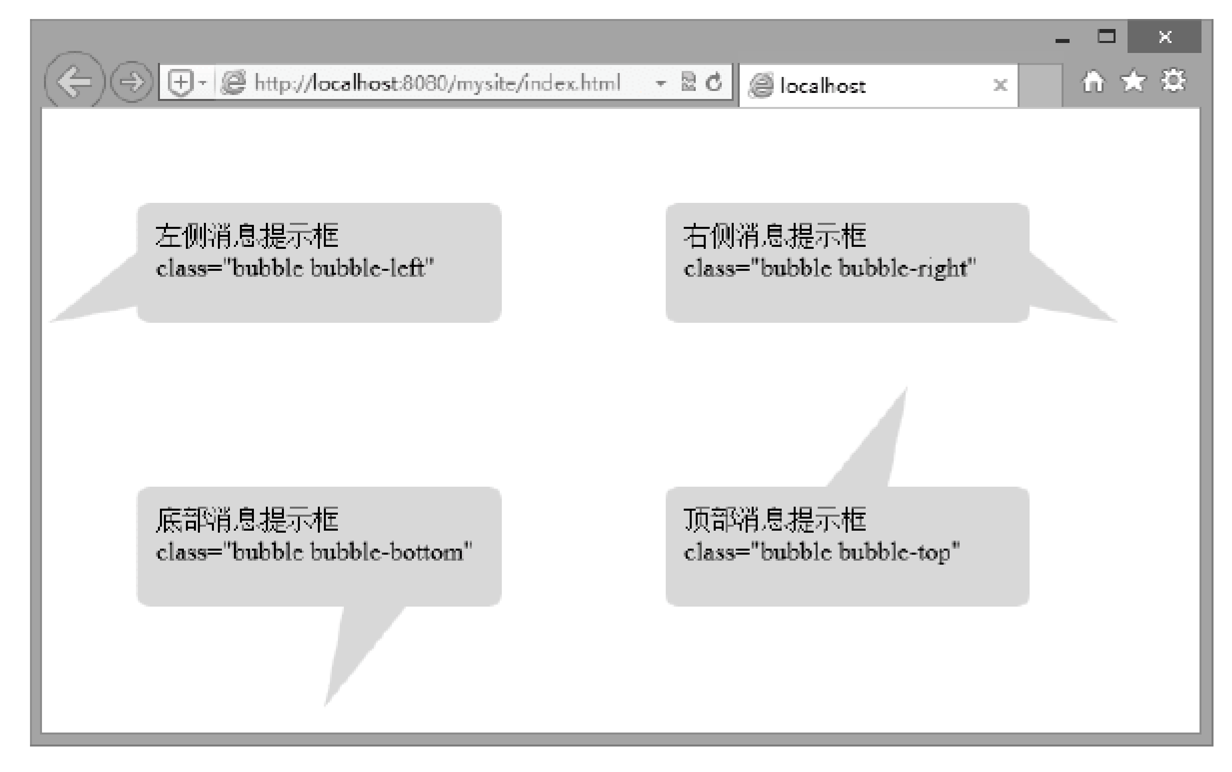


图 20.38　设计消息提示框



线上阅读

具体操作步骤请扫码学习。

20.7　在 线 练 习

练习使用 CSS 设计各种网页文本效果，以及各种网页特效版式和文本，强化基本功训练。



在线练习

第21章

CSS3 背景图像和渐变背景

在 CSS 2.1 中，background 属性的功能还无法满足设计的需求，为了方便设计师更灵活地设计需要的网页效果，CSS3 在原有 background 基础上新增了一些功能属性，可以在同一个对象内叠加多个背景图像，可以改变背景图像的大小尺寸，还可以指定背景图像的显示范围，以及指定背景图像的绘制起点等。另外，CSS3 允许用户使用渐变函数绘制背景图像，这极大地降低了网页设计的难度，激发了设计师的创意灵感。

【学习重点】

⏭ 设置背景图像的原点、大小。

⏭ 正确使用背景图像裁切属性。

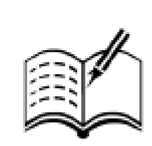
⏭ 灵活使用多重背景图像设计网页版面。

⏭ 正确使用线性渐变和径向渐变。

⏭ 熟练使用渐变函数设计网页元件。



21.1　设计背景图像





CSS3 增强了 background 属性的功能，允许在同一个元素内叠加多个背景图像，还新增了 3 个与背景相关的属性：background-clip、background-origin、background-size。下面分别进行介绍。

权威参考：http://www.w3.org/TR/css3-background/。

21.1.1　设置定位原点

background-origin 属性定义 background-position 属性的定位原点。在默认情况下，background-position 属性总是根据元素左上角为坐标原点进行背景图像定位。使用 background-origin 属性可以改变这种定位方式。该属性的基本语法如下所示：

```
background-origin:border-box | padding-box | content-box;
```

取值简单说明如下：

- ☑ border-box：从边框区域开始显示背景。
- ☑ padding-box：从补白区域开始显示背景，为默认值。
- ☑ content-box：仅在内容区域显示背景。

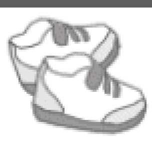
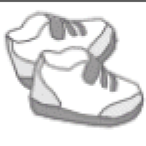
【示例】background-origin 属性改善了背景图像定位的方式，更灵活地决定背景图像应该显示的位置。下面示例利用 background-origin 属性重设背景图像的定位坐标，以便更好地控制背景图像的显示，演示效果如图 21.1 所示。



图 21.1　设计诗词效果

示例代码如下所示：

```
<style type="text/css">
div {/*定义包含框的样式*/
    height: 322px;
```

```
        width: 780px;
        border: solid 1px red;
        padding: 250px 4em 0;
        /*为了避免背景图像重复平铺到边框区域，应禁止它平铺*/
        background:url(images/p3.jpg) no-repeat;
        /*设计背景图像的定位坐标点为元素边框的左上角*/
        background-origin:border-box;
        /*将背景图像等比缩放到完全覆盖包含框，背景图像有可能超出包含框*/
        background-size:cover;
        overflow:hidden;   /*隐藏超出包含框的内容*/
    }
    div h1, div h2{/*定义标题样式*/
        font-size:18px; font-family:"幼圆";
        text-align:center;   /*水平居中显示*/
    }
    div p {/*定义正文样式*/
        text-indent:2em;   /*首行缩进 2 个字符*/
        line-height:2em;   /*增大行高，让正文看起来更疏朗*/
        margin-bottom:2em;     /*调整底部边界，增大段落文本距离*/
    }
    </style>
    <div>
        <h1>念奴娇&#8226;赤壁怀古</h1>
        <h2>苏轼</h2>
        <p>大江东去，浪淘尽，千古风流人物。故垒西边，人道是，三国周郎赤壁。乱石穿空，惊涛拍岸，卷起千堆雪。江山如画，一时多少豪杰。</p>
        <p>遥想公瑾当年，小乔初嫁了，雄姿英发。羽扇纶巾，谈笑间，樯橹灰飞烟灭。故国神游，多情应笑我，早生华发。人生如梦，一尊还酹江月。</p>
    </div>
```



视频讲解

21.1.2 设置裁剪区域

background-clip 属性定义背景图像的裁剪区域。该属性的基本语法如下所示。

```
background-clip:border-box | padding-box | content-box | text;
```

取值简单说明如下：

☑ border-box：从边框区域向外裁剪背景，为默认值。

☑ padding-box：从补白区域向外裁剪背景。

☑ content-box：从内容区域向外裁剪背景。

☑ text：从前景内容（如文字）区域向外裁剪背景。

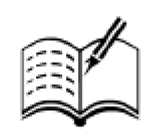
提示：如果取值为 padding-box，则 background-image 将忽略补白边缘，此时边框区域显示为透明；

如果取值为 border-box，则 background-image 将包括边框区域；

如果取值为 content-box，则 background-image 将只包含内容区域；

如果 background-image 属性定义了多重背景，则 background-clip 属性值可以设置多个值，并用逗号分隔。





Note

如果 background-clip 属性值为 padding-box，background-origin 属性取值为 border-box，且 background-position 属性值为"top left"（默认初始值），则背景图左上角将会被截取掉一部分。

【示例 1】下面示例演示如何设计背景图像仅在内容区域内显示，演示效果如图 21.2 所示。

```
<style type="text/css">
div {
    height:150px;
    width:300px;
    border:solid 50px gray;
    padding:50px;
    background:url(images/bg.jpg) no-repeat;
    /*将背景图像等比缩放到完全覆盖包含框，背景图像有可能超出包含框*/
    background-size:cover;
    /*将背景图像从 content 区域开始向外裁剪背景*/
    background-clip:content-box;
}
</style>

<div></div>
```

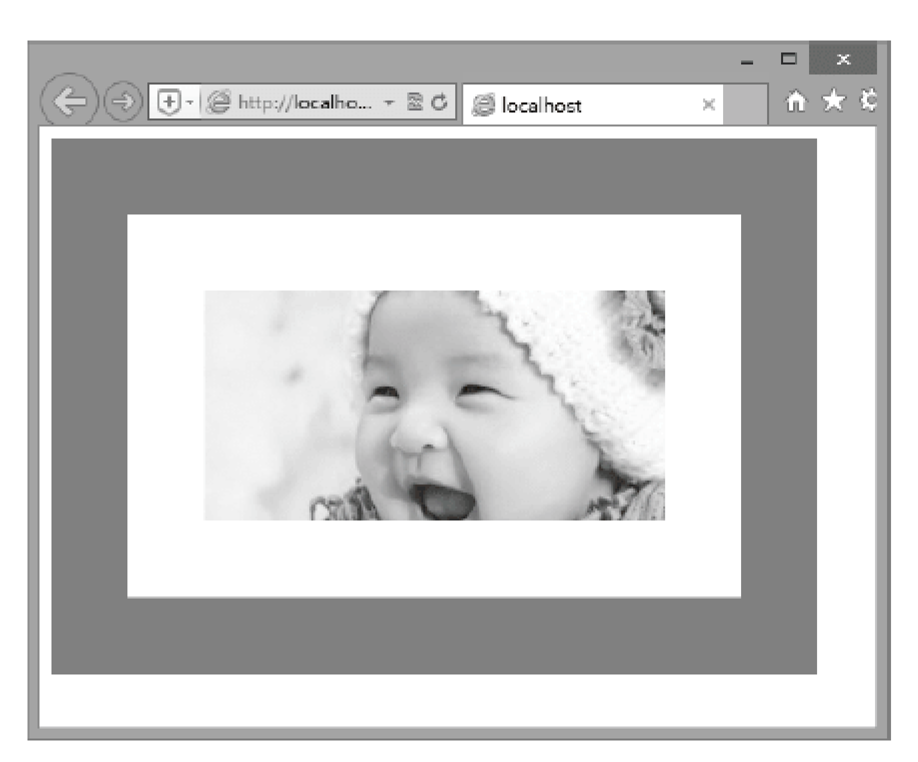


图 21.2　以内容边缘裁切背景图像效果

【示例 2】下面示例同时定义 background-clip 和 background-origin 属性值为 content，可以设计比较特殊的按钮样式，演示效果如图 21.3 所示。

```
<style type="text/css">
button {
    height:40px;        /*固定包含框大小*/
    width:150px;
    padding:1px;        /*在内容区留点空隙*/
    cursor:pointer;     /*定义手形指针样式*/
    color:#fff;         /*白色字体*/
    /*设计立体边框样式*/
    border:3px double #95071b;
    border-right-color:#650513;
    border-bottom-color:#650513;
    /*为了避免背景图像重复平铺到边框区域，应禁止它平铺 */
    background:url(images/img6.jpg) no-repeat;
```



```
        /*设计背景图像的定位坐标点为元素内容区域的左上角*/
        background-origin:content-box;
        /*设计背景图像以内容区域的边缘进行裁切背景图像*/
        background-clip:content-box;
    }
    </style>
    <button>导航按钮 >></button>
```

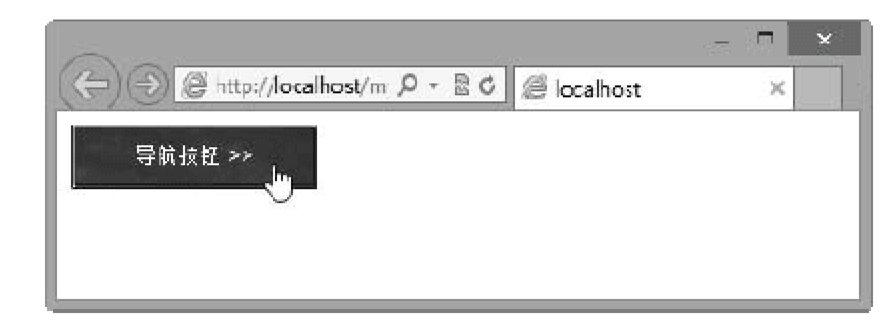


图 21.3　设计按钮效果



视 频 讲 解

21.1.3　设置背景图像大小

background-size 可以控制背景图像的显示大小。该属性的基本语法如下所示：

```
background-size: [ <length> | <percentage> | auto ]{1,2} | cover | contain;
```

取值简单说明如下：

☑　<length>：由浮点数字和单位标识符组成的长度值。不可为负值。

☑　<percentage>：取值为 0 到 100%之间的值。不可为负值。

☑　cover：保持背景图像本身的宽高比例，将图片缩放到正好完全覆盖所定义背景的区域。

☑　contain：保持图像本身的宽高比例，将图片缩放到宽度或高度正好适应所定义背景区域。

初始值为 auto。background-size 属性可以设置 1 个或 2 个值，1 个为必填，1 个为可选。其中第一个值用于指定背景图像的 width，第二个值用于指定背景图像的 height，如果只设置 1 个值，则第二个值默认为 auto。

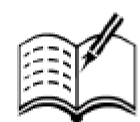
【示例】下面示例使用 image-size 属性自由定制背景图像的大小，让背景图像自适应盒子的大小，从而可以设计与模块大小完全适应的背景图像，本示例效果如图 21.4 所示，只要背景图像长宽比与元素长宽比相同，就不用担心背景图像变形显示。



图 21.4　设计背景图像自适应显示

示例代码如下所示：






```
<style type="text/css">
div {
    margin:2px;
    float:left;
    border:solid 1px red;
    background:url(images/img2.jpg) no-repeat center;
    /*设计背景图像完全覆盖元素区域*/
    background-size:cover;}
/*设计元素大小*/
.h1 { height:80px; width:110px; }
.h2 { height:400px; width:550px; }
</style>
<div class="h1"></div>
<div class="h2"></div>
```

21.1.4　设置多重背景图像



视频讲解

CSS3 支持在同一个元素内定义多个背景图像，还可以将多个背景图像进行叠加显示，从而使得设计多图背景栏目变得更加容易。

【示例 1】本例使用 CSS3 多背景设计花边框，使用 background-origin 定义仅在内容区域显示背景，使用 background-clip 属性定义背景从边框区域向外裁剪，如图 21.5 所示。



图 21.5　设计花边框效果

示例代码如下所示：

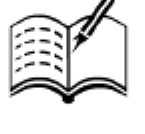
```
<style type="text/css">
.demo {
    /*设计元素大小、补白、边框样式，边框为 20 像素，颜色与背景图像色相同*/
    width: 400px; padding: 30px 30px; border: 20px solid rgba(104, 104, 142,0.5);
    /*定义圆角显示*/
    border-radius: 10px;
    /*定义字体显示样式*/
    color: #f36; font-size: 80px; font-family:"隶书";line-height: 1.5; text-align: center;
}
.multipleBg {
    /*定义 5 个背景图，分别定位到 4 个顶角，其中前 4 个禁止平铺，最后一个可以平铺*/
    background: url("images/bg-tl.png") no-repeat left top,
                url("images/bg-tr.png") no-repeat right top,
                url("images/bg-bl.png") no-repeat left bottom,
```

```
            url("images/bg-br.png") no-repeat right bottom,
            url("images/bg-repeat.png") repeat left top;
    /*改变背景图像的 position 原点，四朵花都是 border 原点，而平铺背景是 padding 原点*/
    background-origin: border-box, border-box, border-box, border-box, padding-box;
    /*控制背景图像的显示区域，所有背景图像超过 border 外边缘都将被剪切掉*/
    background-clip: border-box;
}
</style>
<div class="demo multipleBg">恭喜发财</div>
```

【**示例 2**】在下面示例中利用 CSS3 多背景图功能设计圆角栏目，效果如图 21.6 所示。

```
<style type="text/css">
.roundbox {
    padding: 2em;
    /*为容器定义 8 个背景图像*/
    background-image: url(images/roundbox1/tl.gif),
                      url(images/roundbox1/tr.gif),
                      url(images/roundbox1/bl.gif),
                      url(images/roundbox1/br.gif),
                      url(images/roundbox1/right.gif),
                      url(images/roundbox1/left.gif),
                      url(images/roundbox1/top.gif),
                      url(images/roundbox1/bottom.gif);
    /*定义 4 个顶角图像禁止平铺，4 个边框图像分别沿 x 轴或 y 轴平铺*/
    background-repeat: no-repeat,
                       no-repeat,
                       no-repeat,
                       no-repeat,
                       repeat-y,
                       repeat-y,
                       repeat-x,
                       repeat-x;
    /*定义 4 个顶角图像分别固定在四个顶角位置，4 个边框图像分别固定在四边位置*/
    background-position: left 0px,
                         right 0px,
                         left bottom,
                         right bottom,
                         right 0px,
                         0px 0px,
                         left 0px,
                         left bottom;
    background-color: #66CC33;
}
</style>
<div class="roundbox">
    <h1>念奴娇&#8226;赤壁怀古</h1>
    <h2>苏轼</h2>
    <p>大江东去，浪淘尽，千古风流人物。故垒西边，人道是，三国周郎赤壁。乱石穿空，惊涛拍岸，卷
起千堆雪。江山如画，一时多少豪杰。</p>
    <p>遥想公瑾当年，小乔初嫁了，雄姿英发。羽扇纶巾，谈笑间，樯橹灰飞烟灭。故国神游，多情应笑
```





```
我，早生华发。人生如梦，一尊还酹江月。</p>
    </div>
```



图 21.6　定义多背景图像



示例效果

注意：每幅背景图像的源、定位坐标以及平铺方式的先后顺序要一一对应。

提示：上面示例用到了多个背景属性：background-image、background-repeat 和 background-position。这些属性都是 CSS 1 中就有的属性，但是在 CSS3 中，允许同时指定多个属性值，多个属性值以逗号作为分隔符，用来指定多个背景图像的显示性质。

21.2　设计渐变背景

W3C 于 2010 年 11 月份正式支持渐变背景样式，该草案作为图像值和图像替换内容模块的一部分进行发布。主要包括 linear-gradient()、radial-gradient()、repeating-linear-gradient() 和 repeating-radial-gradient()四个渐变函数。

权威参考：http://dev.w3.org/csswg/css3-images/#gradients。

权威参考

21.2.1　定义线性渐变



视频讲解

创建一个线性渐变，至少需要两个颜色，也可以选择设置一个起点或一个方向。简明语法格式如下：

```
linear-gradient( angle, color-stop1, color-stop2, …)
```

参数简单说明如下：

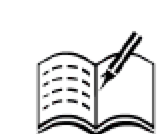
☑ angle：用来指定渐变的方向，可以使用角度或者关键字来设置。关键字包括 4 个，说明如下。

- to left：设置渐变为从右到左，相当于 270deg。
- to right：设置渐变从左到右，相当于 90deg
- to top：设置渐变从下到上，相当于 0deg
- to bottom：设置渐变从上到下，相当于 180deg。该值为默认值。

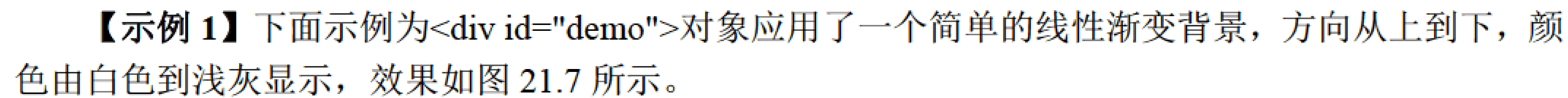
提示：如果创建对角线渐变，可以使用 to top left（从右下到左上）类似组合来实现。



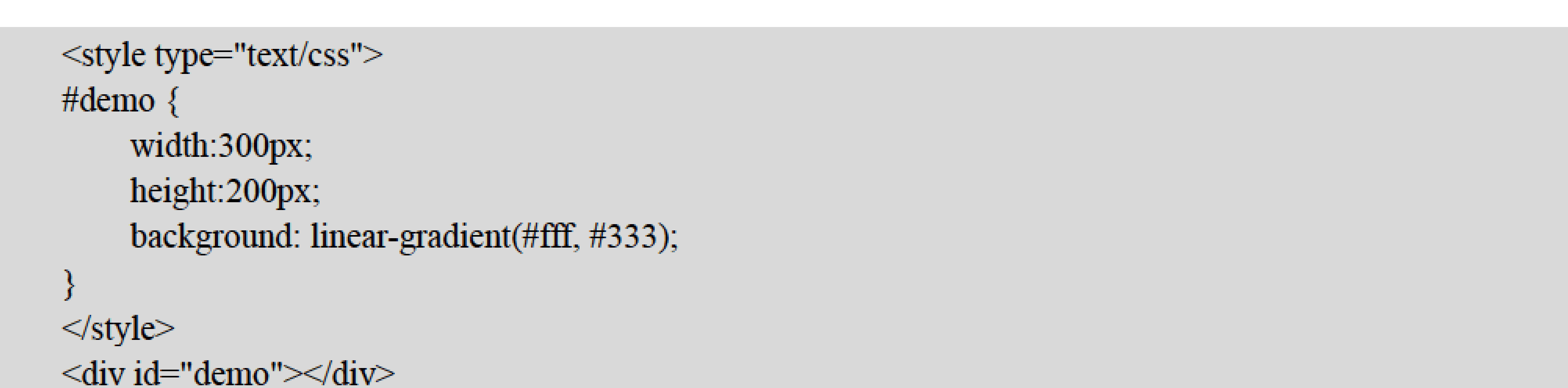
☑ color-stop：用于指定渐变的色点。包括一个颜色值和一个起点位置，颜色值和起点位置以空格分隔。起点位置可以为一个具体的长度值（不可为负值）；也可以是一个百分比值，如果是百分比值则参考应用渐变对象的尺寸，最终会被转换为具体的长度值。



Note



【示例 1】下面示例为<div id="demo">对象应用了一个简单的线性渐变背景，方向从上到下，颜色由白色到浅灰显示，效果如图 21.7 所示。



```
<style type="text/css">
#demo {
    width:300px;
    height:200px;
    background: linear-gradient(#fff, #333);
}
</style>
<div id="demo"></div>
```

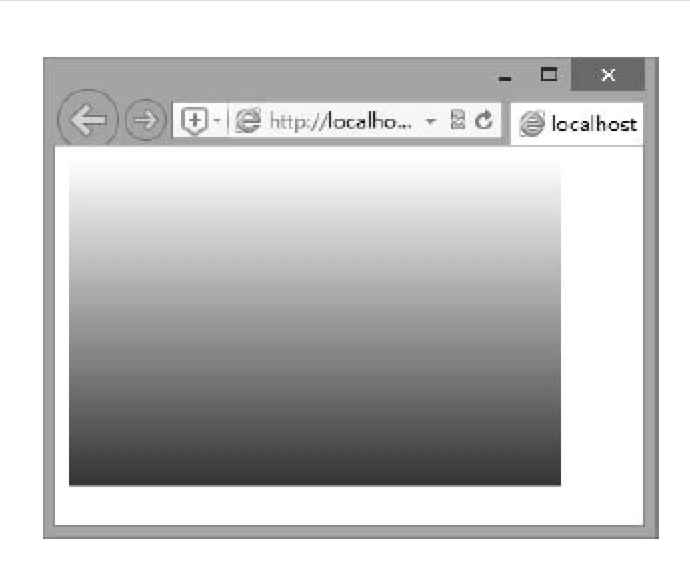


图 21.7　应用简单的线性渐变效果

提示：针对示例 1，用户可以继续尝试做下面练习，实现不同的设置，得到相同的设计效果。

☑ 设置一个方向：从上到下，覆盖默认值。

```
linear-gradient(to bottom, #fff, #333);
```

☑ 设置反向渐变：从下到上，同时调整起止颜色位置。

```
linear-gradient(to top, #333, #fff);
```

☑ 使用角度值设置方向。

```
linear-gradient(180deg, #fff, #333);
```

☑ 明确起止颜色的具体位置，覆盖默认值。

```
linear-gradient(to bottom, #fff 0%, #333 100%);
```

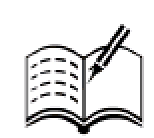
提示：最新主流浏览器都支持线性渐变的标准用法，但是考虑到安全性，用户应酌情兼容旧版本浏览器的私有属性。

Webkit 是第一个支持渐变的浏览器引擎（Safari 4+），它使用-webkit-gradient()私有函数支持线性渐变样式，简明用法如下：

```
-webkit-gradient(linear, point, point, stop)
```

参数简单说明如下：





☑ linear：定义渐变类型为线性渐变。

☑ point：定义渐变起始点和结束点坐标。该参数支持数值、百分比和关键字，如(0 0)或者(left top)等。关键字包括 top、bottom、left 和 right。

☑ stop：定义渐变色和步长。包括三个值，即开始的颜色，使用 from(colorvalue)函数定义；结束的颜色，使用 to(colorvalue)函数定义；颜色步长，使用 color-stop(value, color value)定义。color-stop()函数包含两个参数值，第一个参数值为一个数值或者百分比值，取值范围为 0～1.0（或者 0～100%），第二个参数值表示任意颜色值。

【示例 2】下面示例针对示例 1，兼容早期 Webkit 引擎的线性渐变实现方法。

```
#demo {
    width:300px; height:200px;
    background: -webkit-gradient(linear, left top, left bottom, from(#fff), to(#333));
    background: linear-gradient(#fff, #333);
}
```

上面示例定义线性渐变背景色，从顶部到底部，从白色向浅灰色渐变显示。

另外，Webkit 引擎也支持-webkit-linear-gradient()私有函数来设计线性渐变。该函数用法与标准函数 linear-gradient()语法格式基本相同。

Firefox 浏览器从 3.6 版本开始支持渐变，Gecko 引擎定义了-moz-linear-gradient()私有函数来设计线性渐变。该函数用法与标准函数 linear-gradient()语法格式基本相同。唯一的区别就是，当使用关键字设置渐变方向时，不带 to 关键字前缀，关键字语义取反。例如，从上到下应用渐变，标准关键字为 to bottom，Firefox 私有属性可以为 top。

【示例 3】下面示例针对示例 1，兼容早期 Gecko 引擎的线性渐变实现方法。

```
#demo {
    width:300px; height:200px;
    background: -webkit-gradient(linear, left top, left bottom, from(#fff), to(#333));
    background: -moz-linear-gradient(top, #fff, #333);
    background: linear-gradient(#fff, #333);
}
```

21.2.2　设计线性渐变样式

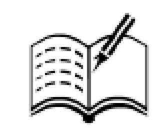


本节以案例形式介绍线性渐变中渐变方向和色点的设置，演示设计线性渐变的一般方法。

【示例 1】下面示例演示了从左边开始的线性渐变。起点是红色，慢慢过渡到蓝色，效果如图 21.8 所示。

```
<style type="text/css">
#demo {
    width:300px; height:200px;
    background: -webkit-linear-gradient(left, red , blue);  /* Safari 5.1 - 6.0 */
    background: -o-linear-gradient(left, red, blue);        /* Opera 11.1 - 12.0 */
    background: -moz-linear-gradient(left, red, blue);      /* Firefox 3.6 - 15 */
    background: linear-gradient(to right, red , blue);      /*  标准语法  */
}
</style>
<div id="demo"></div>
```





注意：第一个参数值渐变方向的设置不同。

【示例 2】通过指定水平和垂直的起始位置来设计对角渐变。下面示例演示了从左上角开始，到右下角的线性渐变，起点是红色，慢慢过渡到蓝色，效果如图 21.9 所示。

```
#demo {
    width:300px; height:200px;
    background: -webkit-linear-gradient(left top, red , blue);    /* Safari 5.1 - 6.0 */
    background: -o-linear-gradient(left top, red, blue);          /* Opera 11.1 - 12.0 */
    background: -moz-linear-gradient(left top, red, blue);        /* Firefox 3.6 - 15 */
    background: linear-gradient(to bottom right, red , blue);     /* 标准语法 */
}
```

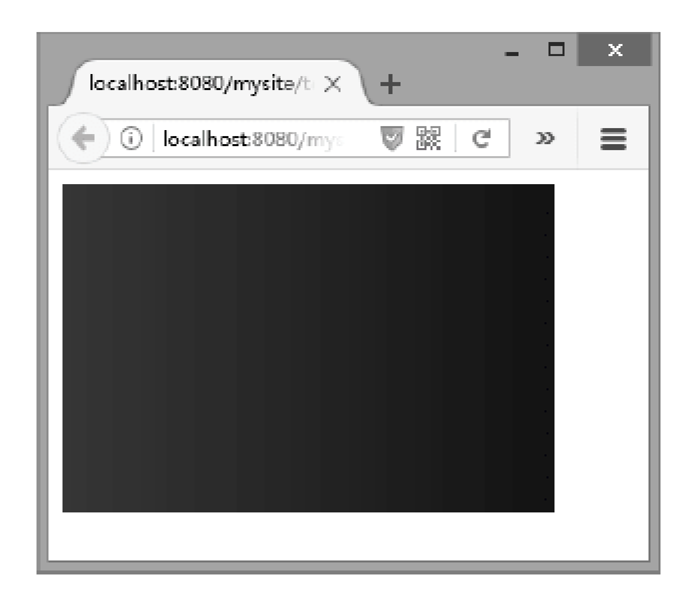


图 21.8　设计从左到右的线性渐变效果

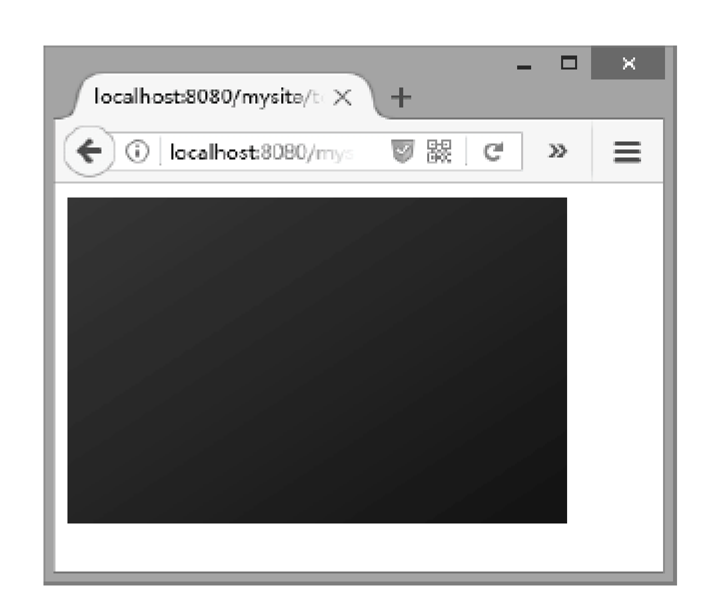


图 21.9　设计对角线性渐变效果

【示例 3】通过指定具体的角度值，可以设计更多渐变方向。下面示例演示了从上到下的线性渐变，起点是红色，慢慢过渡到蓝色，效果如图 21.10 所示。

```
#demo {
    width:300px; height:200px;
    background: -webkit-linear-gradient(-90deg, red, blue);    /* Safari 5.1 - 6.0 */
    background: -o-linear-gradient(-90deg, red, blue);         /* Opera 11.1 - 12.0 */
    background: -moz-linear-gradient(-90deg, red, blue);       /* Firefox 3.6 - 15 */
    background: linear-gradient(180deg, red, blue);            /* 标准语法 */
}
```

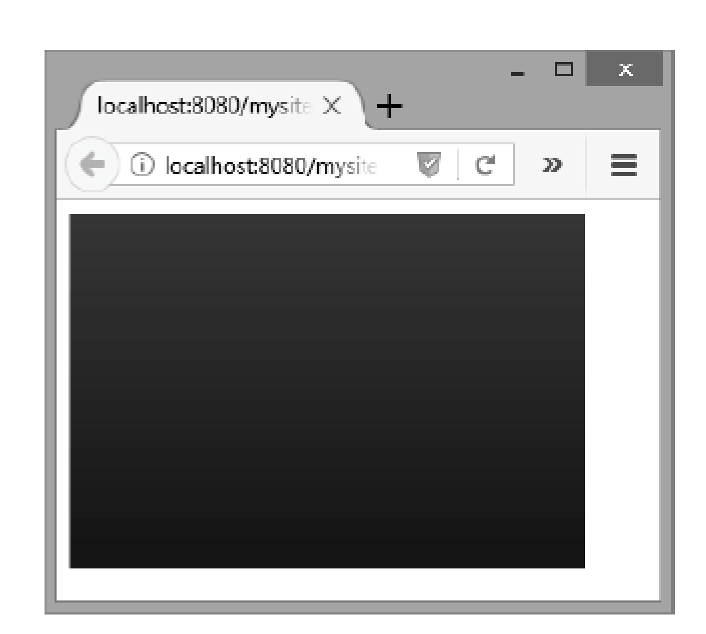
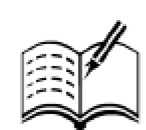


图 21.10　设计从上到下的渐变效果

注意：渐变角度是指垂直线和渐变线之间的角度，逆时针方向计算。例如，0deg 将创建一个从下到上的渐变，90deg 将创建一个从左到右的渐变。注意，渐变起点以负 Y 轴为参考。





但是，很多浏览器（如 Chrome、Safari、Firefox 等）使用旧的标准：渐变角度是指水平线和渐变线之间的角度，逆时针方向计算。例如，0deg 将创建一个从左到右的渐变，90deg 将创建一个从下到上的渐变。注意，渐变起点以负 X 轴为参考。

兼容公式：

```
90 - x = y
```

其中，x 为标准角度，y 为非标准角度。

【示例 4】设置多个色点。下面示例定义从上到下的线性渐变，起点是红色，慢慢过渡到绿色，再慢慢过渡到蓝色，效果如图 21.11 所示。

```
#demo {
    width:300px; height:200px;
    background: -webkit-linear-gradient(red, green, blue);        /* Safari 5.1 - 6.0 */
    background: -o-linear-gradient(red, green, blue);             /* Opera 11.1 - 12.0 */
    background: -moz-linear-gradient(red, green, blue);           /* Firefox 3.6 - 15 */
    background: linear-gradient(red, green, blue);                /*  标准语法  */
}
```

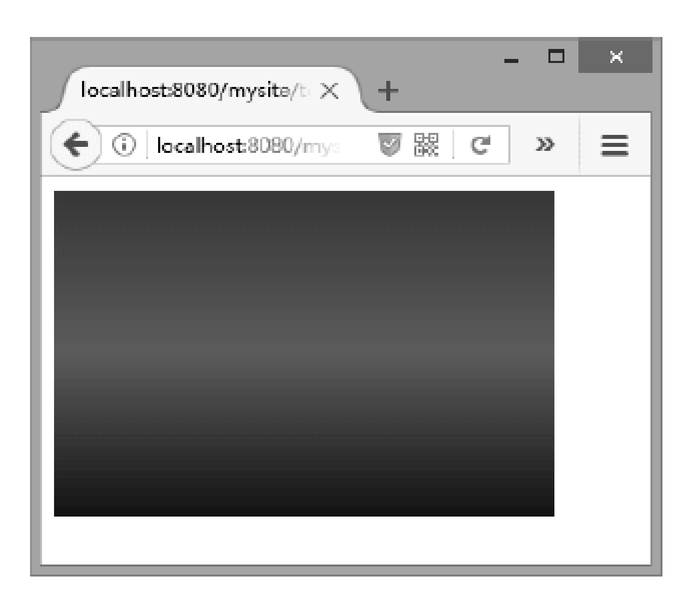


图 21.11　设计多色线性渐变效果

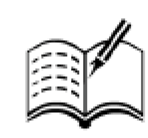
【示例 5】设置色点位置。下面示例定义从上到下的线性渐变，起点是黄色，快速过渡到蓝色，再慢慢过渡到绿色，效果如图 21.12 所示。

```
#demo {
    width:300px; height:200px;
    background: -webkit-linear-gradient(yellow, blue 20%, #0f0);        /* Safari 5.1 - 6.0 */
    background: -o-linear-gradient(yellow, blue 20%, #0f0);             /* Opera 11.1 - 12.0 */
    background: -moz-linear-gradient(yellow, blue 20%, #0f0);           /* Firefox 3.6 - 15 */
    background: linear-gradient(yellow, blue 20%, #0f0);                /*  标准语法  */
}
```

【示例 6】CSS3 渐变支持透明度设置，可用于创建减弱变淡的效果。下面示例演示了从左边开始的线性渐变。起点是完全透明，起点位置为 30%，慢慢过渡到完全不透明的红色，为了更清晰地看到半透明效果，示例增加了一层背景图像进行衬托，演示效果如图 21.13 所示。

```
#demo {
    width:300px; height:200px;
    /* Safari 5.1 - 6 */
    background: -webkit-linear-gradient(left,rgba(255,0,0,0) 30%,rgba(255,0,0,1)),url(images/bg.jpg);
    /* Opera 11.1 - 12*/
    background: -o-linear-gradient(left,rgba(255,0,0,0) 30%,rgba(255,0,0,1)),url(images/bg.jpg);
```





Note

```
        /* Firefox 3.6 - 15*/
        background: -moz-linear-gradient(left,rgba(255,0,0,0) 30%,rgba(255,0,0,1)),url(images/bg.jpg);
        /* 标准语法 */
        background: linear-gradient(to right, rgba(255,0,0,0) 30%, rgba(255,0,0,1)),url(images/bg.jpg);
        background-size:cover;              /* 背景图像完全覆盖 */
}
```

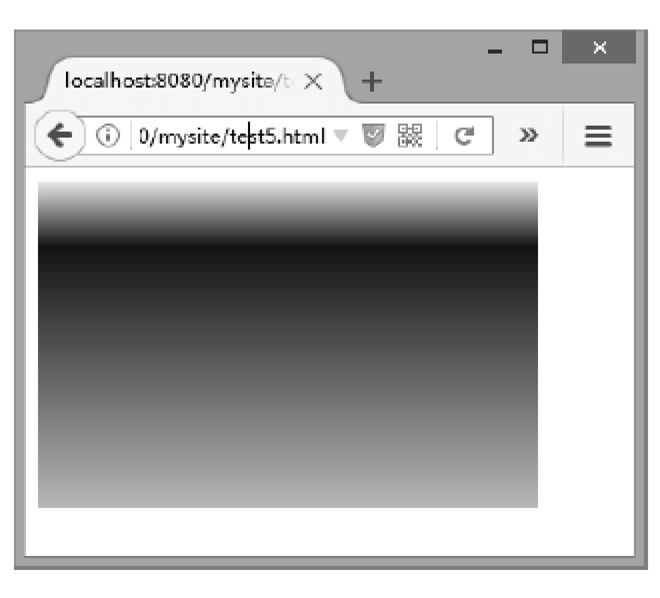


图 21.12　设计多色线性渐变效果

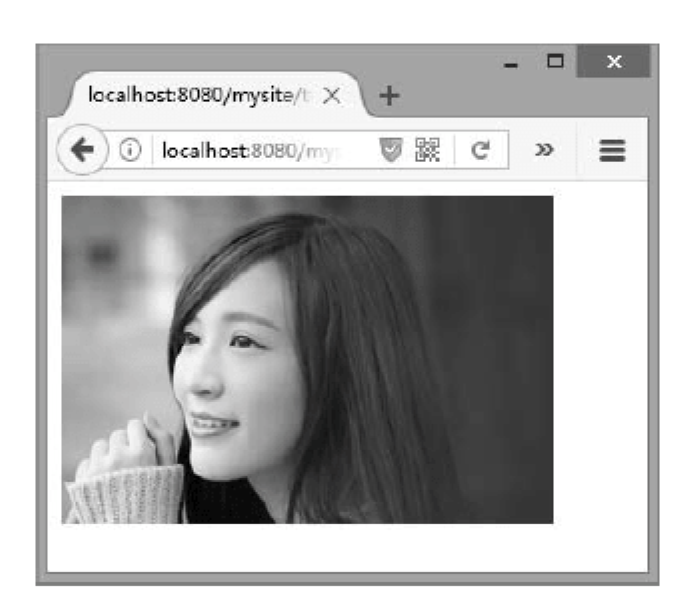


图 21.13　设计半透明线性渐变效果

提示： 为了添加透明度，可以使用 rgba() 或 hsla()函数来定义色点。rgba()或 hsla()函数中最后一个参数可以是从 0 到 1 的值，它定义了颜色的透明度：0 表示完全透明，1 表示完全不透明。

21.2.3　案例：设计网页渐变色



视频讲解

为页面设计渐变背景，可以营造特殊的浏览气氛。本例主要代码如下所示：

```
<style type="text/css">
body { /*让渐变背景填满整个页面*/
    padding: 1em;
    margin: 0;
    background: -webkit-linear-gradient(#FF6666, #ffffff);        /* Safari 5.1 - 6.0 */
    background: -o-linear-gradient(#FF6666, #ffffff);             /* Opera 11.1 - 12.0 */
    background: -moz-linear-gradient(#FF6666, #ffffff);           /* Firefox 3.6 - 15 */
    background: linear-gradient(#FF6666, #ffffff);                /* 标准语法 */
    /* IE 滤镜，兼容 IE9-版本浏览器 */
 filter: progid:DXImageTransform.Microsoft.Gradient(gradientType=0, startColorStr=#FF6666, endColorStr=#ffffff);
}
h1 {/* 定义标题样式 */
    color: white;
    font-size: 18px;
    height: 45px;
    padding-left: 3em;
    line-height: 50px; /* 控制文本显示位置 */
    border-bottom: solid 2px red;
    background: url(images/pe1.png) no-repeat left center; /*为标题插入一个装饰图标*/
}
p { text-indent: 2em; }/* 段落文本缩进 2 个字符 */
</style>
<div  class="box">
    <h1>W3C 发布 HTML5 的正式推荐标准</h1>
```



```
    <p>2014 年 10 月 28 日，W3C 的 HTML 工作组正式发布了 HTML5 的正式推荐标准（W3C Recommendation）。W3C 在美国圣克拉拉举行的 W3C 技术大会及顾问委员会会议（TPAC 2014）上宣布了这一消息。 HTML5 是万维网的核心语言－可扩展标记语言的第 5 版。在这一版本中，增加了支持 Web 应用开发者的许多新特性，以及更符合开发者使用习惯的新元素，并重点关注定义清晰的、一致的准则，以确保 Web 应用和内容在不同用户代理（浏览器）中的互操作性。HTML5 是构建开放 Web 平台的核心。</p>
    <p class="right">更多<a href="http://www.chinaw3c.org/archives/677/" target="_blank">详细内容</a></p>
</div>
```



Note

预览效果如图 21.14 所示。





示 例 效 果

图 21.14　设计渐变网页背景色效果

【补充】

IE 早期版本不支持 CSS 渐变，但提供了渐变滤镜，可以实现简单的渐变效果。IE 浏览器渐变滤镜的基本语法说明如下。

```
filter:progid:DXImageTransform.Microsoft.Gradient(enabled=bEnabled,startColorStr=iWidth,endColorStr=iWidth)
```

该函数的参数说明如下：

☑ enabled：设置或检索滤镜是否激活。可选布尔值，包括 true 和 false，默认值为 true，激活状态。

☑ startColorStr：设置或检索色彩渐变的开始颜色和透明度。可选项，其格式为#AARRGGBB。AA、RR、GG、BB 为十六进制正整数，取值范围为 00~FF。RR 指定红色值，GG 指定绿色值，BB 指定蓝色值。AA 指定透明度，00 是完全透明，FF 是完全不透明。超出取值范围的值将被恢复为默认值。取值范围为#FF000000～#FFFFFFFF，默认值为#FF0000FF，即不透明蓝色。

☑ endColorStr：设置或检索色彩渐变的结束颜色和透明度。默认值为#FF000000，即不透明黑色。

注意： IE 渐变滤镜在 IE 5.5 及以上版本浏览器中有效。

21.2.4　案例：设计条纹背景



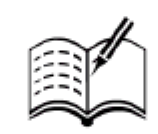
视 频 讲 解

如果多个色点设置相同的起点位置，它们将产生一个从一种颜色到另一种颜色的急剧的转换。从效果来看，就是从一种颜色突然改变到另一种颜色，这样可以设计条纹背景效果。

【示例 1】 定义一个简单的条纹背景，效果如图 21.15 所示。

```
<style type="text/css">
#demo {
    height:200px;
```





Note

```
        background: linear-gradient(#cd6600 50%, #0067cd 50%);
    }
    </style>
    <div id="demo"></div>
```

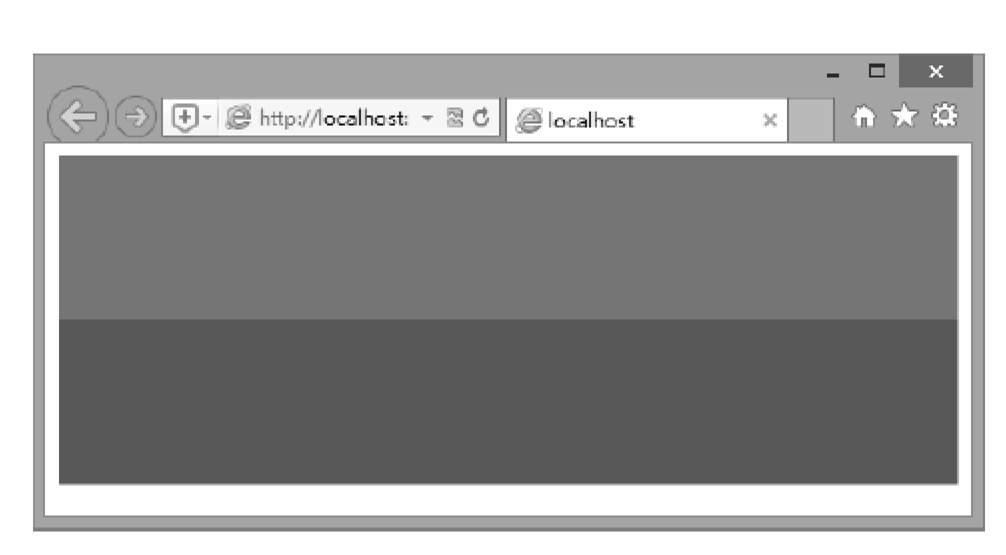


图 21.15　设计简单的条纹效果

【示例 2】利用背景的重复机制，可以创造出更多的条纹。示例代码如下所示：

```
#demo {
    height:200px;
    background: linear-gradient(#cd6600 50%, #0067cd 50%);
    background-size: 100% 20%;  /*  定义单个条纹仅显示高度的五分之一  */
}
```

效果如图 21.16 所示。这样就可以将整个背景划分为 10 个条纹，每个条纹的高度一样。



图 21.16　设计重复显示的条纹效果

【示例 3】如果设计每个条纹高度不同，只要改变比例即可，示例代码如下所示：

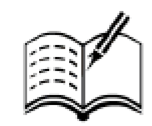
```
#demo {
    height:200px;
    background: linear-gradient(#cd6600 80%, #0067cd 0%);/*定义每个条纹位置占比不同  */
    background-size: 100% 20%;  /*  定义单个条纹仅显示高度的五分之一  */
}
```

效果如图 21.17 所示。

【示例 4】设计多色条纹背景，代码如下所示：

```
#demo {
    height:200px;
    /*  定义三色同宽背景  */
    background: linear-gradient(#cd6600 33.3%, #0067cd 0, #0067cd 66.6%, #00cd66 0);
    background-size: 100% 30px;
}
```





效果如图 21.18 所示。



图 21.17　设计不同高度的条纹效果



图 21.18　设计多色条纹效果

【示例 5】设计密集条纹格效果，代码如下所示：

```
#demo {
    height:200px;
    background: linear-gradient(rgba(0,0,0,.5) 1px, #fff 1px);
    background-size: 100% 3px;
}
```

效果如图 21.19 所示。

注意：IE 不支持这种设计效果。

【示例 6】设计垂直条纹背景，只需要转换一下宽和高的设置方式，具体代码如下所示：

```
#demo {
    height:200px;
    background: linear-gradient(to right, #cd6600 50%, #0067cd 0);
    background-size: 20% 100%;
}
```

效果如图 21.20 所示。



图 21.19　设计密集条纹效果

图 21.20　设计垂直条纹效果

【示例 7】设计简单的纹理背景，代码如下所示：

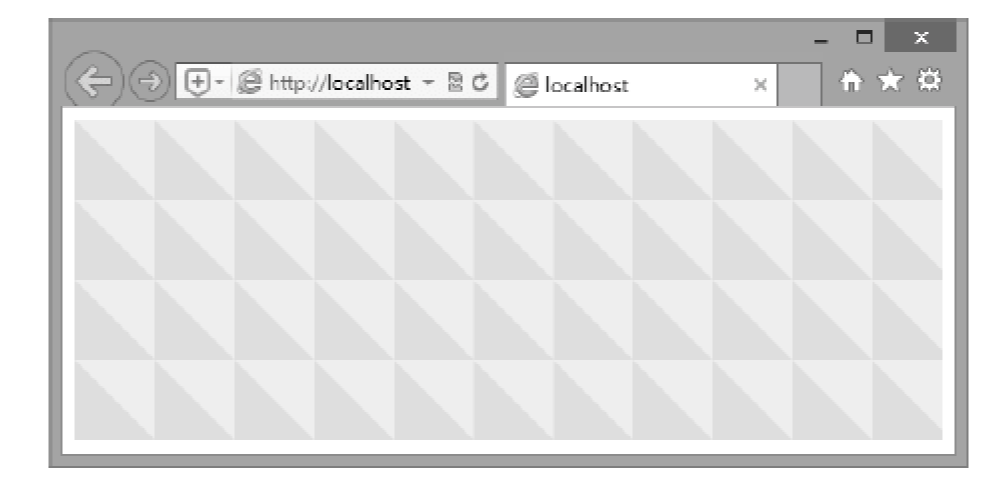
```
#demo {
    height:200px;
    background: linear-gradient(45deg, RGBA(0,103,205,0.2)  50%, RGBA(0,103,205,0.1)  50%);
    background-size: 50px 50px;
}
```

效果如图 21.21 所示。



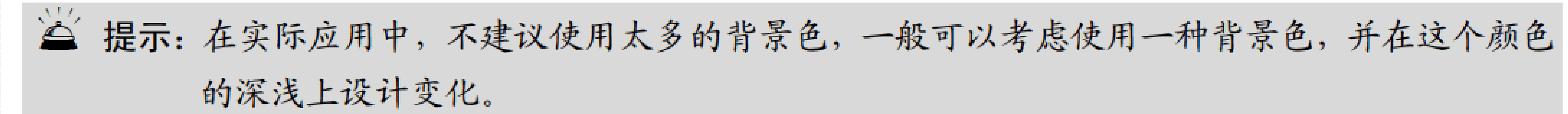


Note



示例效果

图 21.21　设计简单的纹理效果



提示：在实际应用中，不建议使用太多的背景色，一般可以考虑使用一种背景色，并在这个颜色的深浅上设计变化。



视频讲解

21.2.5　定义重复线性渐变

使用 repeating-linear-gradient()函数可以定义重复线性渐变，用法与 linear-gradient()函数相同，用户可以参考 21.2.1 节说明。

提示：使用重复线性渐变的关键是要定义好色点，让最后一个颜色和第一个颜色能够很好地连接起来，处理不当将导致颜色的急剧变化。

【示例 1】下面示例设计重复显示的垂直线性渐变，颜色从红色到蓝色，间距为 20%，效果如图 21.22 所示。

```
<style type="text/css">
#demo {
    height:200px;
    background: repeating-linear-gradient(#f00, #00f 20%, #f00 40%);
}
</style>
<div id="demo"></div>
```

提示：使用 linear-gradient()可以设计 repeating-linear-gradient()的效果，例如，通过重复设计每一个色点，或者利用 21.2.4 节设计条纹方法来实现。

【示例 2】下面示例设计重复线性渐变对角显示，效果如图 21.23 所示。

```
#demo {
    height:200px;
    background: repeating-linear-gradient(135deg, #cd6600, #0067cd 20px, #cd6600 40px);
}
```

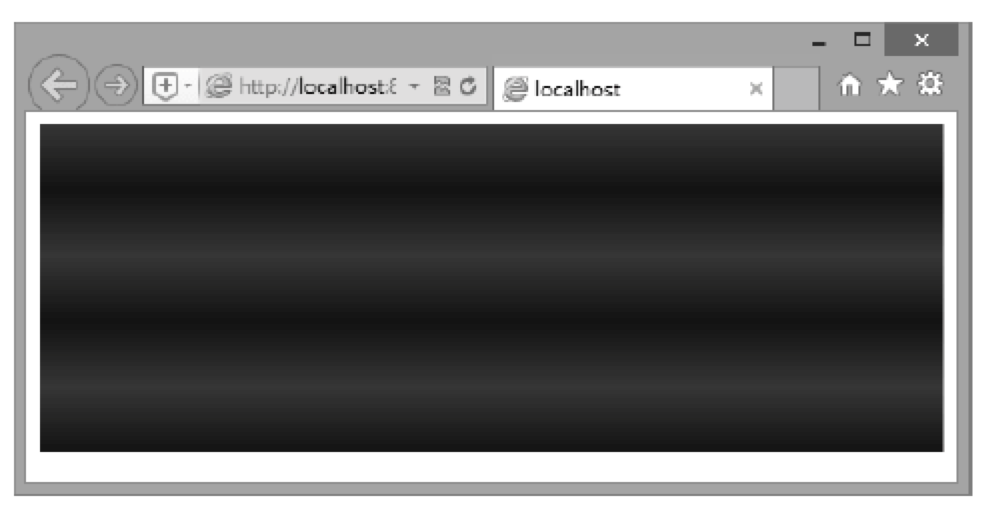


图 21.22　设计重复显示的垂直渐变效果



图 21.23　设计重复显示的对角渐变效果



【示例 3】下面示例设计使用重复线性渐变创建出对角条纹背景，效果如图 21.24 所示。

```
#demo {
    height:200px;
    background: repeating-linear-gradient(60deg, #cd6600, #cd6600 5%, #0067cd 0, #0067cd 10%);
}
```

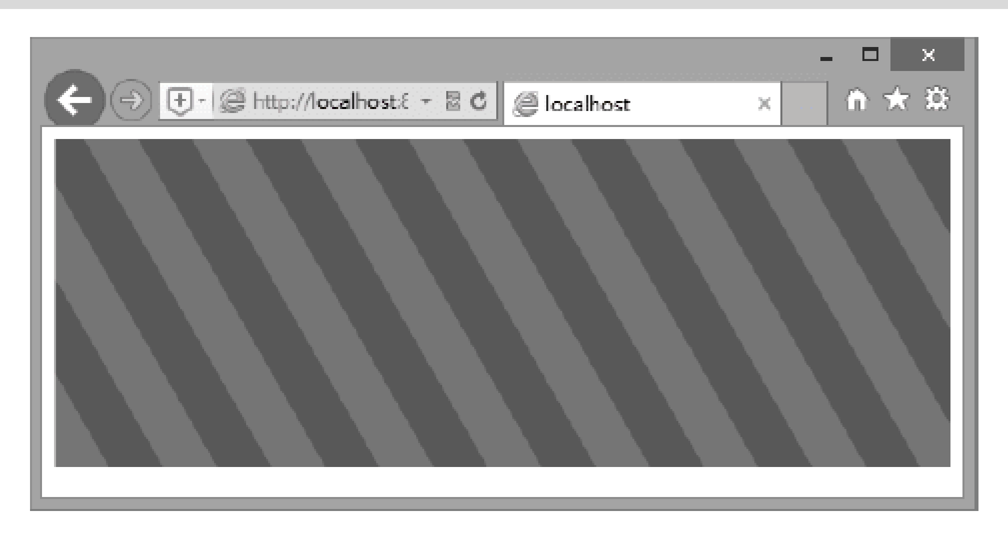
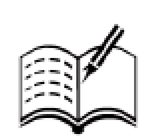


图 21.24　设计重复显示的对角条纹效果



Note

21.2.6　定义径向渐变



视 频 讲 解

创建一个径向渐变，也至少需要定义两个颜色，同时可以指定渐变的中心点位置、形状类型（圆形或椭圆形）和半径大小。简明语法格式如下：

```
radial-gradient(shape size at position, color-stop1, color-stop2, …);
```

参数简单说明如下：

☑ shape：用来指定渐变的类型，包括 circle（圆形）和 ellipse（椭圆）两种。

☑ size：如果类型为 circle，指定一个值设置圆的半径；如果类型为 ellipse，指定两个值分别设置椭圆的 x 轴和 y 轴半径。取值包括长度值、百分比、关键字。关键字说明如下。

- closest-side：指定径向渐变的半径长度为从中心点到最近的边。
- closest-corner：指定径向渐变的半径长度为从中心点到最近的角。
- farthest-side：指定径向渐变的半径长度为从中心点到最远的边。
- farthest-corner：指定径向渐变的半径长度为从中心点到最远的角。

☑ position：用来指定中心点的位置。如果提供两个参数，第一个表示 x 轴坐标，第二个表示 y 轴坐标；如果只提供一个值，第二个参数值默认为 50%，即 center。取值可以是长度值、百分比或者关键字，关键字包括 left（左侧）、center（中心）、right（右侧）、top（顶部）、center（中心）、bottom（底部）。

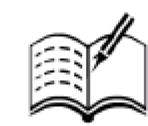
注意：position 值位于 shape 和 size 值后面。

☑ color-stop：用于指定渐变的色点。包括一个颜色值和一个起点位置，颜色值和起点位置以空格分隔。起点位置可以为一个具体的长度值（不可为负值）；也可以是一个百分比值，如果是百分比值，则参考应用渐变对象的尺寸，最终会被转换为具体的长度值。

【示例 1】在默认情况下，渐变的中心是 center（对象中心点），渐变的形状是 ellipse（椭圆形），渐变的大小是 farthest-corner（表示到最远的角落）。下面示例仅为 radial-gradient()函数设置 3 个颜色值，则它将按默认值绘制径向渐变效果，如图 21.25 所示。

```
<style type="text/css">
#demo {
    height:200px;
```





Note

```
      background: -webkit-radial-gradient(red, green, blue);        /* Safari 5.1 - 6.0 */
      background: -o-radial-gradient(red, green, blue);             /* Opera 11.6 - 12.0 */
      background: -moz-radial-gradient(red, green, blue);           /* Firefox 3.6 - 15 */
      background: radial-gradient(red, green, blue);                /* 标准语法 */
}
</style>
<div id="demo"></div>
```

提示：针对示例 1，用户可以继续尝试做下面的练习，实现不同的设置，得到相同的设计效果。

☑ 设置径向渐变形状类型，默认值为 ellipse。

```
background: radial-gradient(ellipse, red, green, blue);
```

☑ 设置径向渐变中心点坐标，默认为对象中心点。

```
background: radial-gradient(ellipse at center 50%, red, green, blue);
```

☑ 设置径向渐变大小，这里定义填充整个对象。

```
background: radial-gradient(farthest-corner, red, green, blue);
```

提示：最新主流浏览器都支持线性渐变的标准用法，但是考虑到安全性，用户应酌情兼容旧版本浏览器的私有属性。

Webkit 引擎使用-webkit-gradient()私有函数支持径向渐变样式，简明用法如下：

```
-webkit-gradient(radial, point, radius, stop)
```

参数简单说明如下：

☑ radial：定义渐变类型为径向渐变。
☑ point：定义渐变中心点坐标。该参数支持数值、百分比和关键字，如(0 0)或者(left top)等。关键字包括 top、bottom、center、left 和 right。
☑ radius：设置径向渐变的长度，该参数为一个数值。
☑ stop：定义渐变色和步长。包括三个值，即开始的颜色，使用 from(colorvalue)函数定义；结束的颜色，使用 to(colorvalue)函数定义；颜色步长，使用 color-stop(value, colorvalue)定义。color-stop()函数包含两个参数值，第一个参数值为一个数值或者百分比值，取值范围在 0 到 1.0 之间（或者 0 到 100%之间），第二个参数值表示任意颜色值。

【示例 2】下面示例设计一个红色圆球，并逐步径向渐变为绿色背景，兼容早期 Webkit 引擎的线性渐变实现方法。代码如下所示：

```
<style type="text/css">
#demo {
      height:200px;
      /* Webkit 引擎私有用法 */
      background: -webkit-gradient(radial, center center, 0, center center, 100, from(red), to(green));
      background: radial-gradient(circle 100px, red, green);    /* 标准的用法 */
}
</style>
<div id="demo"></div>
```



演示效果如图 21.26 所示。

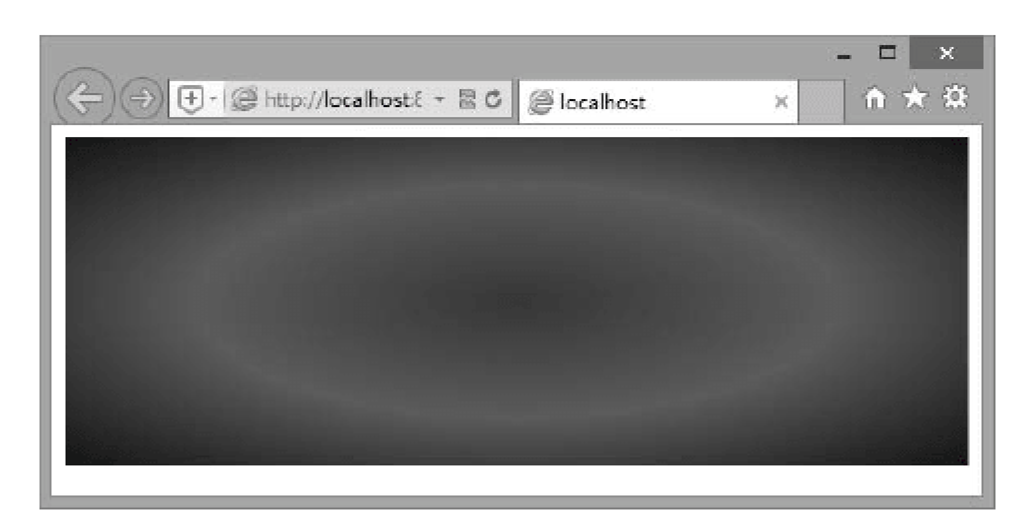


图 21.25　设计简单的径向渐变效果

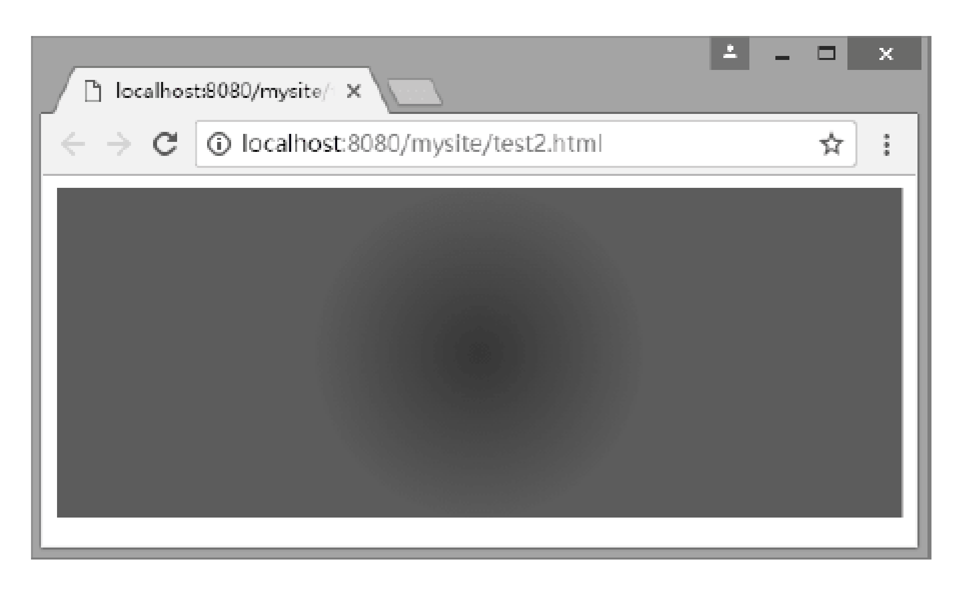
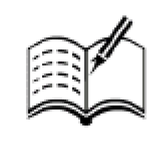


图 21.26　设计径向圆球效果



另外，Webkit 引擎也支持-webkit-radial-gradient()私有函数来设计径向渐变。该函数用法与标准函数 radial-gradient()语法格式类似。简明语法格式如下：

```
-webkit-radial-gradient(position, shape size, color-stop1, color-stop2, …);
```

Gecko 引擎定义了-moz-radial-gradient()私有函数来设计径向渐变。该函数用法与标准函数 radial-gradient()语法格式也类似。简明语法格式如下：

```
-moz-radial-gradient(position, shape size, color-stop1, color-stop2, …);
```



提示：上面两个私有函数的 size 参数值仅可设置关键字：closest-side、closest-corner、farthest-side、farthest-corner、contain 或 cover。

21.2.7　设计径向渐变样式



视 频 讲 解

本节以案例形式介绍径向渐变的灵活设置，熟练掌握设计径向渐变的一般方法。

【示例 1】下面示例演示了色点不均匀分布的径向渐变，效果如图 21.27 所示。

```
<style type="text/css">
#demo {
    height:200px;
    background: -webkit-radial-gradient(red 5%, green 15%, blue 60%);    /* Safari 5.1 - 6.0 */
    background: -o-radial-gradient(red 5%, green 15%, blue 60%);         /* Opera 11.6 - 12.0 */
    background: -moz-radial-gradient(red 5%, green 15%, blue 60%);       /* Firefox 3.6 - 15 */
    background: radial-gradient(red 5%, green 15%, blue 60%);            /*  标准语法  */
}
</style>
<div id="demo"></div>
```

【示例 2】shape 参数定义了形状，取值包括 circle 和 ellipse，其中 circle 表示圆形，ellipse 表示椭圆形，默认值是 ellipse。下面示例设计了圆形径向渐变，效果如图 21.28 所示。

```
#demo {
    height:200px;
    background: -webkit-radial-gradient(circle, red, yellow, green);   /* Safari 5.1 - 6.0 */
    background: -o-radial-gradient(circle, red, yellow, green);        /* Opera 11.6 - 12.0 */
    background: -moz-radial-gradient(circle, red, yellow, green);      /* Firefox 3.6 - 15 */
    background: radial-gradient(circle, red, yellow, green);           /*  标准语法  */
}
```



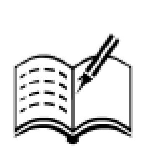
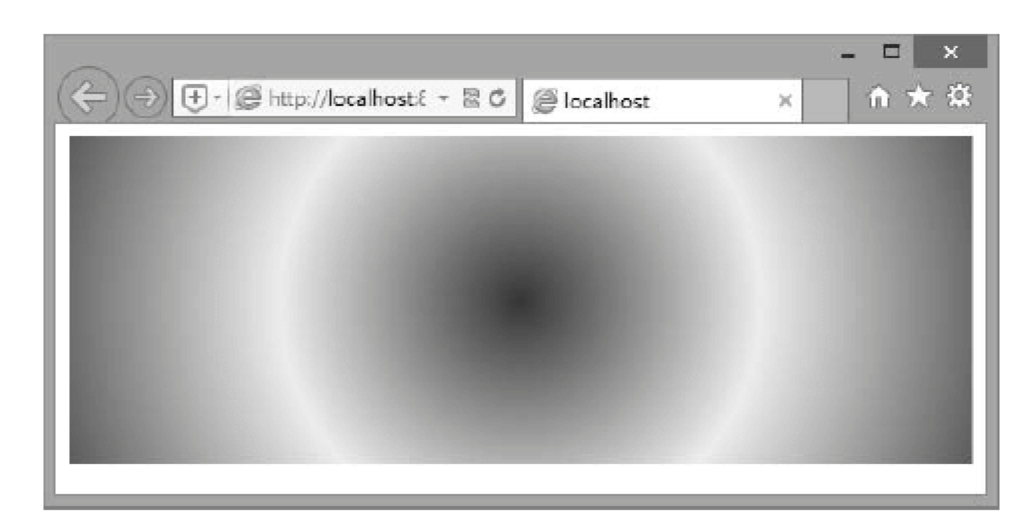

Note


图 21.27　设计色点不均匀分布的径向渐变效果


图 21.28　设计圆形径向渐变效果

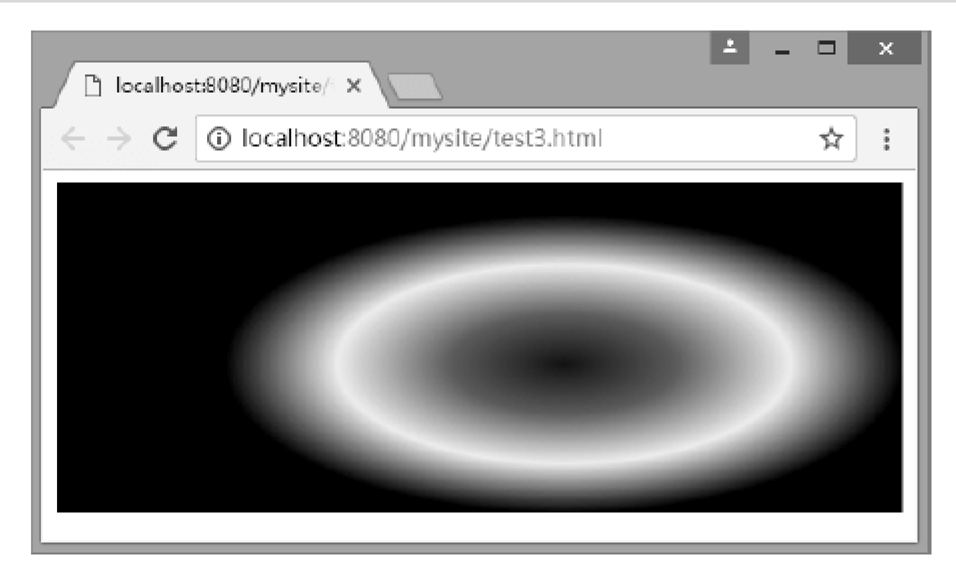
【示例 3】下面设计径向渐变的半径长度为从圆心到离圆心最近的边，效果如图 21.29 所示。

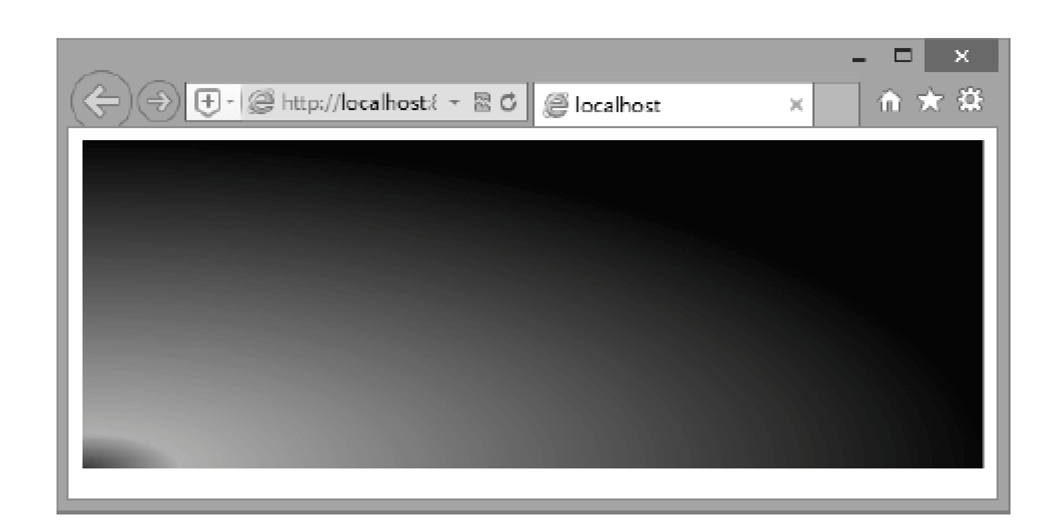
```
#demo {
    height:200px;
    /* Safari 5.1 - 6.0 */
    background: -webkit-radial-gradient(60% 55%, closest-side,blue,green,yellow,black);
    /* Opera 11.6 - 12.0 */
    background: -o-radial-gradient(60% 55%, closest-side,blue,green,yellow,black);
    /* Firefox 3.6 - 15 */
    background: -moz-radial-gradient(60% 55%, closest-side,blue,green,yellow,black);
    /*  标准语法  */
    background: radial-gradient(closest-side at 60% 55%, blue,green,yellow,black);
}
```

注意：radial-gradient()标准函数与各私有函数在设置参数时存在顺序区别。

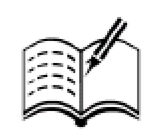
【示例 4】下面示例模拟太阳初升的效果，如图 21.30 所示。设计径向渐变中心点位于左下角，半径为最大化显示，定义 3 个色点，第一个色点设计太阳效果，第二个色点设计太阳余晖，第三个色点设计太空色，第一个色点和第二个色点距离为 60 像素。

```
#demo {
    height:200px;
    /* Safari 5.1 - 6.0 */
    background: -webkit-radial-gradient(left bottom, farthest-side, #f00, #f99 60px, #005);
    /* Opera 11.6 - 12.0 */
    background: -o-radial-gradient(left bottom, farthest-side, #f00, #f99 60px, #005);
    /* Firefox 3.6 - 15 */
    background: -moz-radial-gradient(left bottom, farthest-side, #f00, #f99 60px, #005);
    /*  标准语法  */
    background: radial-gradient(farthest-side at left bottom, #f00, #f99 60px, #005);
}
```


图 21.29　设计最小限度的径向渐变效果


图 21.30　模拟太阳初升效果





Note

【示例 5】下面示例模拟太阳旗效果，如图 21.31 所示。设计径向渐变中心点位于对象中央，定义两个色点，第一个色点设计太阳效果，第二个色点设计背景，两个色点位置相同。

```
<style type="text/css">
body { background:hsla(207,59%,78%,1.00) }
#demo {
    height:200px;
    width:300px;
    margin:auto;
    /* Safari 5.1 - 6.0 */
    background: -webkit-radial-gradient(center, circle, #f00 50px, #fff 50px);
    /* Opera 11.6 - 12.0 */
    background: -o-radial-gradient(center, circle, #f00 50px, #fff 50px);
    /* Firefox 3.6 - 15 */
    background: -moz-radial-gradient(center, circle, #f00 50px, #fff 50px);
    /*  标准语法  */
    background: radial-gradient(circle   at center, #f00 50px, #fff 50px);
}
</style>
<div id="demo"></div>
```

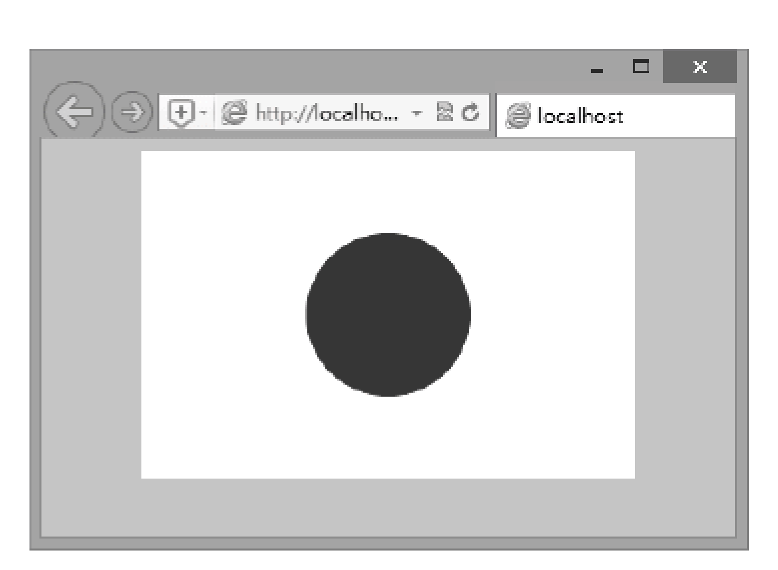


图 21.31　设计太阳旗效果

21.2.8　定义重复径向渐变



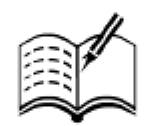
使用 repeating-radial-gradient()函数可以定义重复线性渐变，用法与 radial-gradient()函数相同，用户可以参考上面说明。

【示例 1】下面示例设计三色重复显示的径向渐变，效果如图 21.32 所示。

```
<style type="text/css">
#demo {
    height:200px;
    /* Safari 5.1 - 6.0 */
    background: -webkit-repeating-radial-gradient(red, yellow 10%, green 15%);
    /* Opera 11.6 - 12.0 */
    background: -o-repeating-radial-gradient(red, yellow 10%, green 15%);
    /* Firefox 3.6 - 15 */
    background: -moz-repeating-radial-gradient(red, yellow 10%, green 15%);
    /*  标准语法  */
    background: repeating-radial-gradient(red, yellow 10%, green 15%);
}
```



```
</style>
<div id="demo"></div>
```



Note

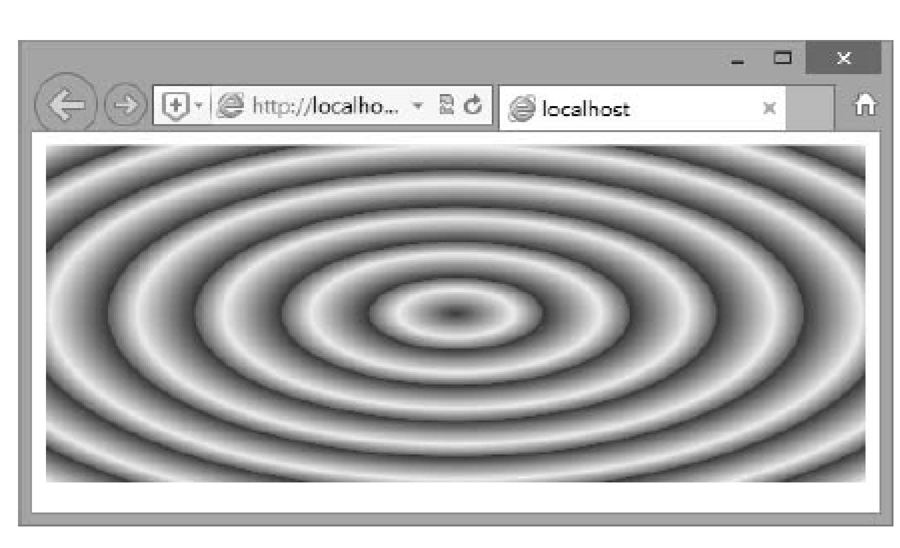


图 21.32　设计重复显示的径向渐变效果

【示例 2】使用径向渐变同样可以创建条纹背景，方法与线性渐变类似。下面示例设计圆形径向渐变条纹背景，效果如图 21.33 所示。

```
#demo {
    height:200px;
    /* Safari 5.1 - 6.0 */
    background: -webkit-repeating-radial-gradient(center bottom,  circle, #00a340, #00a340 20px, #d8ffe7 20px, #d8ffe7 40px);
    /* Opera 11.6 - 12.0 */
    background: -o-repeating-radial-gradient(center bottom, circle,  #00a340, #00a340 20px, #d8ffe7 20px, #d8ffe7 40px);
    /* Firefox 3.6 - 15 */
    background: -moz-repeating-radial-gradient(center bottom, circle, #00a340, #00a340 20px, #d8ffe7 20px, #d8ffe7 40px);
    /*  标准语法  */
    background: repeating-radial-gradient(circle at center bottom, #00a340, #00a340 20px, #d8ffe7 20px, #d8ffe7 40px);
}
```



图 21.33　设计径向渐变条纹背景效果



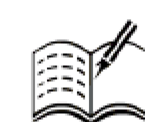
视 频 讲 解

21.2.9　案例：设计网页背景色

【示例 1】为页面叠加多个径向渐变背景，可以营造虚幻的页面氛围。示例代码如下所示：

```
<style type="text/css">
html, body{ height:100%;}
body {
```





Note

```
        background-color: #4B770A;
        background-image:
            radial-gradient( rgba(255, 255, 255, 0.3), rgba(255, 255, 255, 0)),
            radial-gradient(at 10% 5%, rgba(255, 255, 255, 0.1), rgba(255, 255, 255, 0) 20%),
            radial-gradient(at left bottom , rgba(255, 255, 255, 0.2), rgba(255, 255, 255, 0) 20%),
            radial-gradient(at right top, rgba(255, 255, 255, 0.2), rgba(255, 255, 255, 0) 20%),
            radial-gradient(at 85% 90% , rgba(255, 255, 255, 0.1), rgba(255, 255, 255, 0) 20%);
    }
    </style>
```

预览效果如图 21.34 所示。

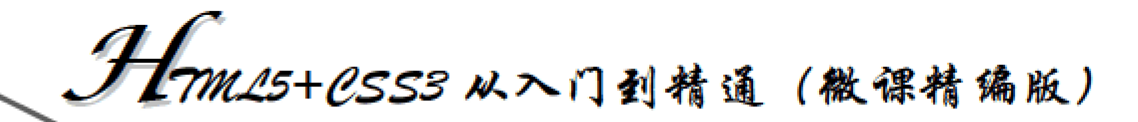


图 21.34　设计多个径向渐变背景效果

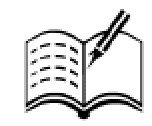
在上面示例代码中，首先设计 body 高度满屏显示，避免无内容时看不到效果；然后为页面定义一个基本色#4B770A；再设计 5 个径向渐变，分别散布于页面四个顶角，以及中央位置，同时定义径向渐变的第一个颜色为半透明的白色，第二个颜色为透明色，从而在页面不同位置蒙上轻重不一的白粉效果，以此来模拟虚幻莫测的背景效果。

【示例 2】为页面叠加 4 个径向渐变背景，设计密密麻麻的针脚纹理效果。示例代码如下所示：

```
<style type="text/css">
html, body{ height:100%;}
body {
    background-color: #282828;
    background-image:
        -webkit-radial-gradient(black 15%, transparent 16%),
        -webkit-radial-gradient(black 15%, transparent 16%),
        -webkit-radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%),
        -webkit-radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%);
    background-image:
        radial-gradient(black 15%, transparent 16%),
        radial-gradient(black 15%, transparent 16%),
        radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%),
        radial-gradient(rgba(255, 255, 255, 0.1) 15%, transparent 20%);
    background-position:
        0 0px,
        8px 8px,
        0 1px,
        8px 9px;
    background-size: 16px 16px;
```

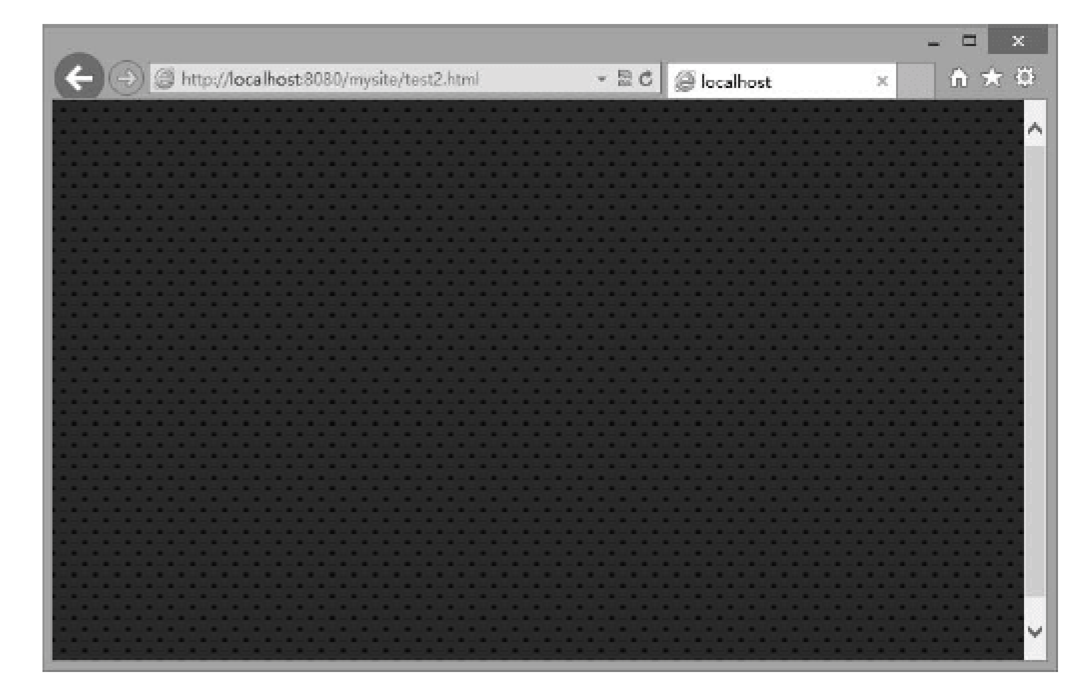





Note

```
}
</style>
```

预览效果如图 21.35 所示。



示例效果

图 21.35　设计针脚纹理背景效果

在上面示例中，首先使用“background-size: 16px 16px;”定义背景图大小为 16×16 像素；在这块小图上设计 4 个径向渐变，包括两个深色径向渐变和两个浅色径向渐变；使用“background-position: 0 0px, 8px 8px, 0 1px, 8px 9px;”设计一深、一浅径向渐变错位叠加，在 y 轴上错位 1 个像素，从而在 16×16 大小的浅色背景图上设计两个深色凹陷效果；最后，借助背景图平铺，为网页设计上述纹理特效。


视频讲解

21.2.10　案例：设计图标

本例通过 CSS3 径向渐变制作圆形图标特效，设计效果如图 21.36 所示。在内部样式表中，使用 radial-gradient()函数为图标标签定义径向渐变背景，设计立体效果；使用“border-radius:50%;”声明定义图标显示为圆形；使用 box-shadow 属性为图标添加投影；使用 text-shadow 属性为图标文本定义润边效果；使用 radial-gradient 设计环形径向渐变效果，为图标添加高亮特效。


示例效果

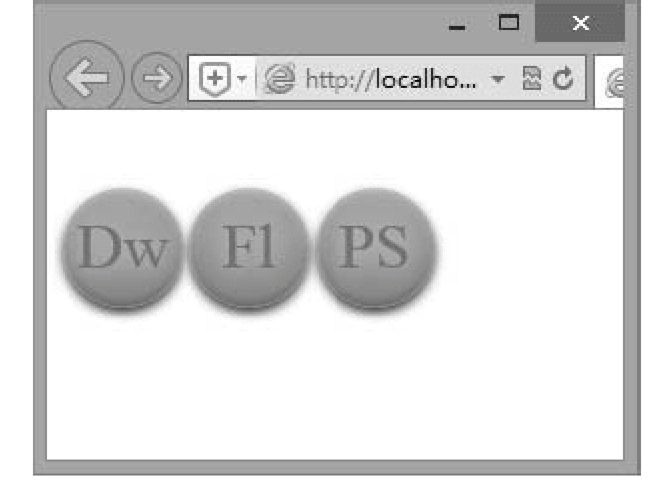


图 21.36　设计径向渐变图标效果

示例主要代码如下：

```
<style type="text/css">
.icon {
    /*  固定大小，可根据实际需要酌情调整，调整时应同步调整 line-height:60px; */
    width: 60px; height: 60px;
    /*  行内块显示，统一图标显示属性  */
    display:inline-block;
    /*  清除边框，避免边框对整体特效的破坏  */
```



```
        border: none;
        /* 设计圆形效果 */
        border-radius: 50%;
        /* 定义图标阴影，第一个外阴影设计立体效果，第二个内阴影设计高亮特效 */
        box-shadow: 0 1px 5px rgba(255,255,255,.5) inset,
                    0 -2px 5px rgba(0,0,0,.3) inset, 0 3px 8px rgba(0,0,0,.8);
        /* 定义径向渐变，模拟明暗变化的表面效果 */
        background: -webkit-radial-gradient( circle at top center, #f28fb8, #e982ad, #ec568c);
        background: radial-gradient(circle at top center, #f28fb8, #e982ad, #ec568c);
        /* 定义图标字体样式 */
        font-size: 32px;
        color: #dd5183;
        text-align:center;      /* 文本水平居中显示 */
        line-height:60px;     /* 文本垂直居中显示，必须与 height: 60px;保持一致 */
        /* 为文本添加阴影，第一个阴影设计立体效果，第二个阴影定义高亮特效 */
        text-shadow: 0 3px 10px #f1a2c1,
                    0 -3px 10px #f1a2c1;
    }
    </style>
    <div class="icon">Dw</div>
    <span class="icon">Fl</span>
    <p class="icon">PS</p>
```



Note

21.3 案例实战

本节将通过多个较复杂案例练习背景样式的实际应用。

21.3.1 设计优惠券



视频讲解

本例使用径向渐变设计一张优惠券效果，如图 21.37 所示。



图 21.37　设计优惠券效果



线上阅读

具体代码解析请扫码学习。

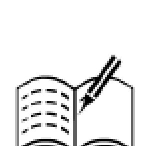
21.3.2 设计桌面纹理背景

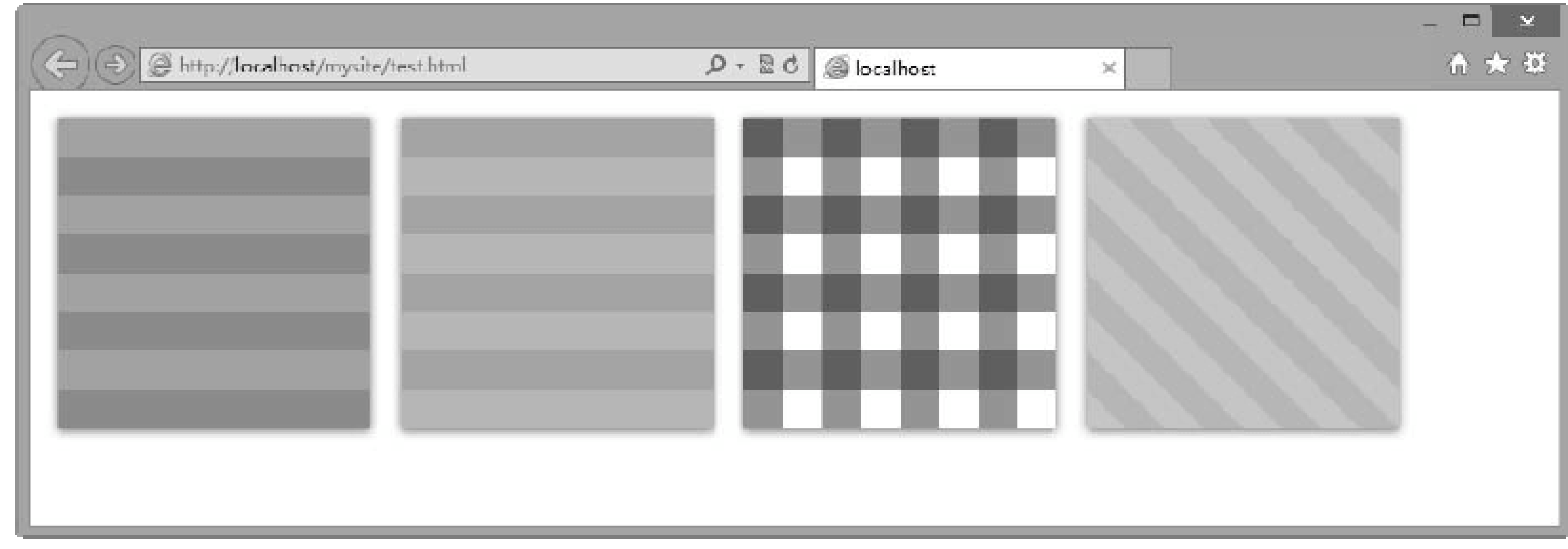


视频讲解

本例使用 CSS3 线性渐变属性制作纹理图案，主要利用多重背景进行设计，然后使用线性渐变绘制每个小方块的线条效果，通过叠加和平铺，完成重复性纹理背景效果，如图 21.38 所示。







线上阅读

图 21.38　定义网页纹理背景效果

具体代码解析请扫码学习。



视频讲解

21.3.3　设计按钮

本节提供了两个经典案例，分别介绍了不同风格的按钮样式设计，效果如图 21.39 和图 21.40 所示。



（a）默认从上到下渐变　　（b）鼠标经过从下到上渐变　　（c）激活时下移 1 像素

图 21.39　设计按钮效果



线上阅读

图 21.40　设计精致的按钮

具体代码解析请扫码学习。

视频讲解

21.3.4　渐变特殊应用场景

渐变可以用在包括 border-image-source、background-image、list-style-image、cursor 等属性上，用来取代 url 属性值。前面各节主要针对 background-image 属性进行介绍，下面结合示例介绍其他属性的应用情形。



线上阅读

☑ 定义渐变效果的边框。
☑ 定义填充内容效果。
☑ 定义列表图标。

详细代码解析请扫码学习。





视频讲解

Note

21.3.5　设计栏目折角效果

灵活使用 CSS3 渐变背景，可以创新出很多新颖的设计。例如，设计缺角效果，如图 21.41 和图 21.42 所示。



图 21.41　设计缺角栏目效果



图 21.42　设计补角栏目效果

设计折边效果，如图 21.43 和图 21.44 所示。

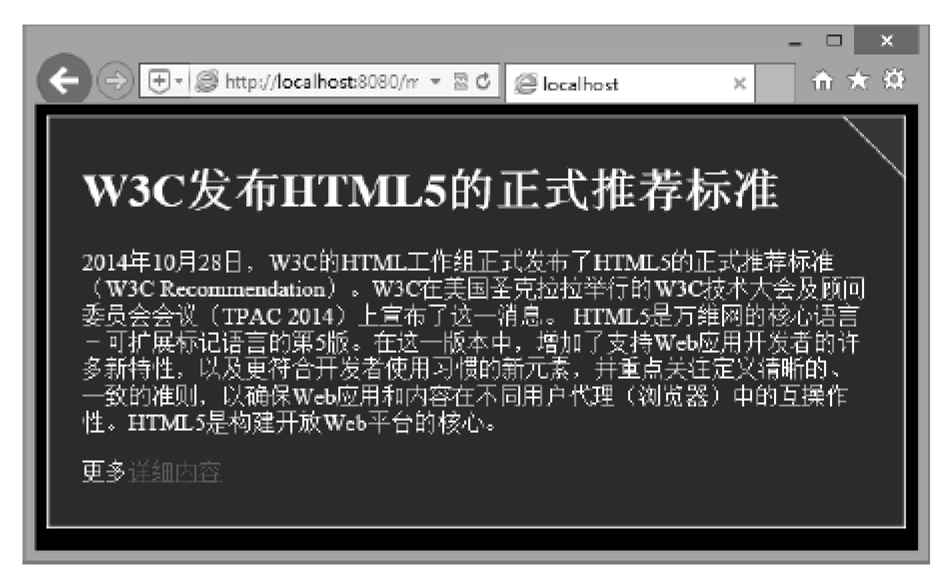


图 21.43　设计高亮边框栏目效果



图 21.44　设计缺角边框栏目效果



线上阅读

详细代码解析请扫码学习。

21.4　在线练习



在线练习

练习使用 CSS3 设计各种网页图像效果，以及各种网页背景图像特效，强化基本功训练。

第22章

CSS3 用户接口样式

2015年4月，W3C的CSS工作组发布CSS基本用户接口模块（CSS Basic User Interface Module Level 3，CSS3 UI）的标准工作草案。该文档描述了CSS3中对HTML、XML进行样式处理所需的与用户界面相关的CSS选择器、属性及属性值。该模块负责控制与用户接口界面相关效果的呈现方式，它包含并扩展了在CSS 2及Selector规范中定义的与用户接口有关的特性。

权威参考：http://www.w3.org/TR/css-ui-3/

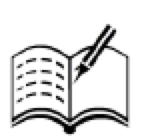


权威参考

【学习重点】

- 了解常用界面显示属性。
- 能够定义轮廓样式。
- 正确设计边框图像样式。
- 灵活设计圆角样式。
- 灵活设计阴影样式。







视频讲解

22.1　界 面 显 示

下面介绍 CSS3 用户界面的显示方式、显示大小和溢出处理问题。

22.1.1　显示方式

一般浏览器都支持两种显示模式：怪异模式和标准模式。在怪异模式下，border 和 padding 包含在 width 或 height 之内；在标准模式下，border、padding、width 或 height 是各自独立区域。

为了兼顾这两种解析模式，CSS3 定义了 box-sizing 属性，该属性能够定义对象尺寸的解析方式。box-sizing 属性的基本语法如下所示。

```
box-sizing : content-box | border-box;
```

取值简单说明如下：

☑ content-box：为默认值，padding 和 border 不被包含在定义的 width 和 height 之内。对象的实际宽度等于设置的 width 值和 border、padding 之和，即元素的宽度 = width + border + padding。

☑ border-box：padding 和 border 被包含在定义的 width 和 height 之内。对象的实际宽度就等于设置的 width 值，即使定义有 border 和 padding 也不会改变对象的实际宽度，即元素的宽度 = width。

【示例】下面示例设计两个相同样式的盒子，在怪异模式和标准模式下比较显示效果，如图 22.1 所示。

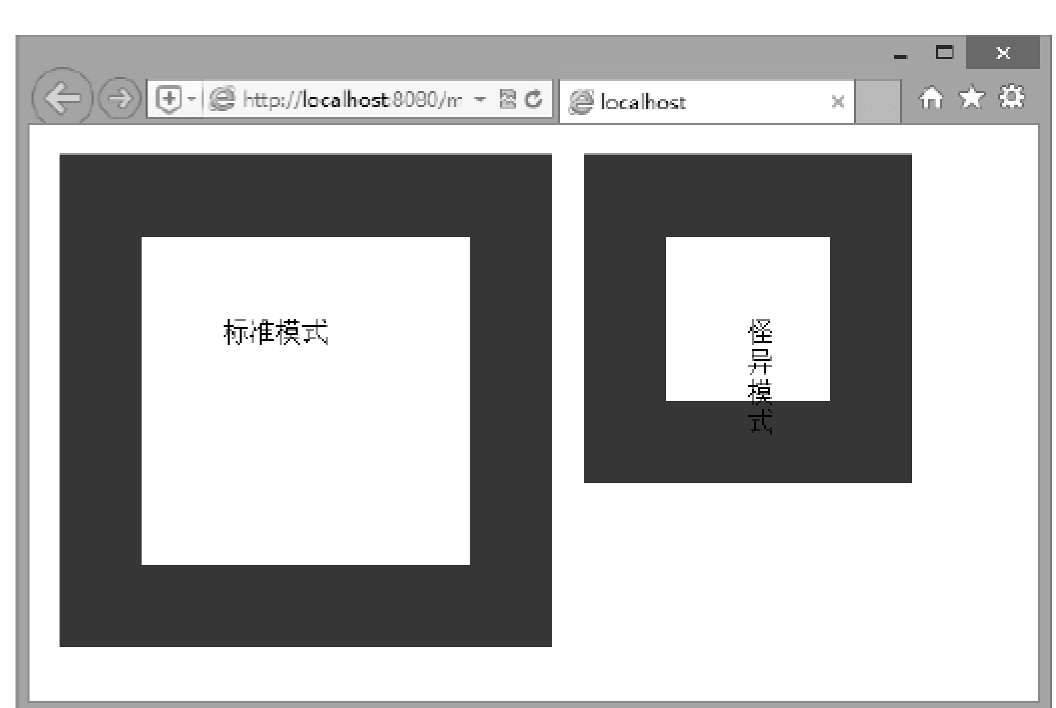


图 22.1　标准模式和怪异模式解析比较

从图 22.1 可以看到，在怪异模式下 width 属性值就是指元素的实际宽度，即 width 属性值中包含 padding 和 border 属性值。

```
<style type="text/css">
div {
    float: left;                    /* 并列显示 */
    height: 100px;                  /* 元素的高度 */
    width: 100px;                   /* 元素的宽度 */
    border: 50px solid red;         /* 边框 */
```



```
        margin: 10px;                    /* 外边距 */
        padding: 50px;                   /* 内边距 */
}
.border-box { box-sizing: border-box;}   /* 怪异模式解析 */
</style>
<div>标准模式</div>
<div class="border-box">怪异模式</div>
```



Note



视频讲解

22.1.2　调整尺寸

为了增强用户体验，CSS3 增加了 resize 属性，允许用户通过拖动的方式改变元素的尺寸。resize 属性的基本语法如下所示：

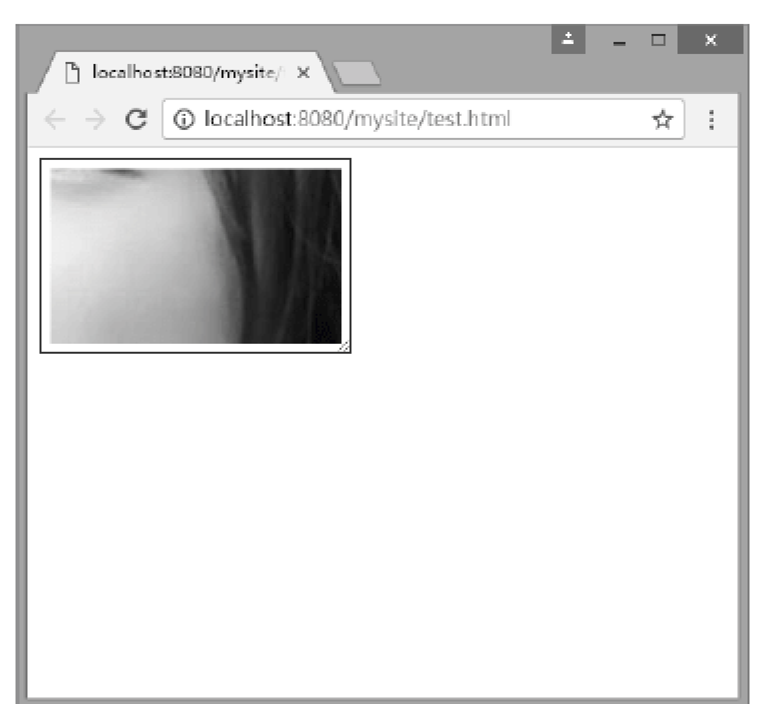
```
resize:none | both | horizontal | vertical | inherit;
```

取值简单说明如下：

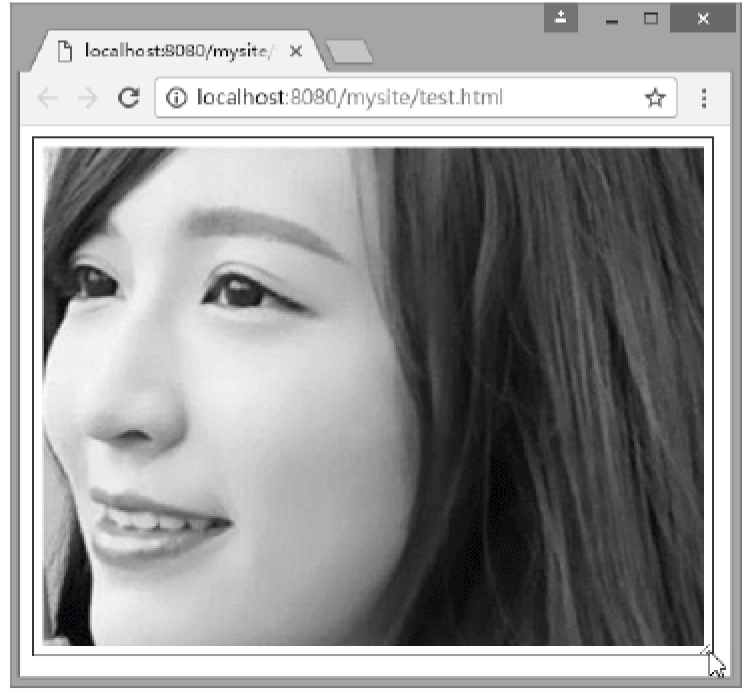
- ☑ none：为默认值，不允许用户调整元素大小。
- ☑ both：用户可以调节元素的宽度和高度。
- ☑ horizontal：用户可以调节元素的宽度。
- ☑ vertical：用户可以调节元素的高度。

目前除了 IE 浏览器外，其他主流浏览器都基本支持该属性。

【示例】下面示例演示如何使用 resize 属性设计自由调整大小的图片，如图 22.2 所示。



（a）默认大小



（b）鼠标拖动放大

图 22.2　调节元素尺寸

```
<style type="text/css">
#resize {
     /*以背景方式显示图像，这样可以更轻松地控制缩放操作*/
    background:url(images/1.jpg) no-repeat center;
     /*设计背景图像仅在内容区域显示，留出补白区域*/
    background-clip:content;
     /*设计元素最小和最大显示尺寸，用户也只能够在该范围内自由调整*/
    width:200px; height:120px;
    max-width:800px; max-height:600px;
    padding:6px; border: 1px solid red;
```



```
        /*必须同时定义 overflow 和 resize，否则 resize 属性声明无效，元素默认溢出显示为 visible*/
        resize: both;
        overflow: auto;
}
</style>
<div id="resize"></div>
```



Note

视频讲解

22.1.3　缩放比例

zoom 是 IE 的专有属性，用于设置对象的缩放比例，另外它还可以触发 IE 的 haslayout 属性，清除浮动，清除 margin 重叠等，设计师常用这个属性解决 IE 浏览器存在的布局 Bug。

CSS3 支持该属性，基本语法如下所示：

```
Zoom:normal | <number> | <percentage>
```

取值说明如下：

☑ normal：使用对象的实际尺寸。

☑ <number>：用浮点数来定义缩放比例，不允许负值。

☑ <percentage>：用百分比来定义缩放比例，不允许负值。

目前，除了 Firefox 浏览器之外，所有主流浏览器都支持该属性。

【示例】 下面示例使用 zoom 放大第 2 幅图片为原来的 2 倍，比较效果如图 22.3 所示。

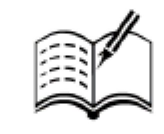
```
<style type="text/css">
img {
    height: 200px;
    margin-right: 6px;
}
img.zoom { zoom: 2; }
</style>
<img src="images/bg.jpg"/>
<img class="zoom" src="images/bg.jpg"/>
```



图 22.3　放大图片显示尺寸

当 zoom 属性值为 1.0 或 100%时，相当于 normal，表示不缩放。小于 1 的正数，表示缩小，如“zoom: 0.5;” 表示缩小一倍。

22.2 轮廓样式



Note



视频讲解

轮廓与边框不同，它不占用空间，且不一定是矩形。轮廓属于动态样式，只有当对象获取焦点或者被激活时呈现，如在按钮、活动窗体域、图形地图等周围添加一圈轮廓线，使对象突出显示。

22.2.1 定义轮廓

outline 属性可以定义块元素的轮廓线，该属性在 CSS 2.1 规范中已被明确定义，但是并未得到各主流浏览器的广泛支持，CSS3 增强了该特性。outline 属性的基本语法如下所示：

```
outline:<'outline-width'> || <'outline-style'> || <'outline-color'> || <'outline-offset'>
```

取值简单说明如下：

- ☑ <'outline-width'>：指定轮廓边框的宽度。
- ☑ <'outline-style'>：指定轮廓边框的样式。
- ☑ <'outline-color'>：指定轮廓边框的颜色。
- ☑ <'outline-offset'>：指定轮廓边框偏移值。

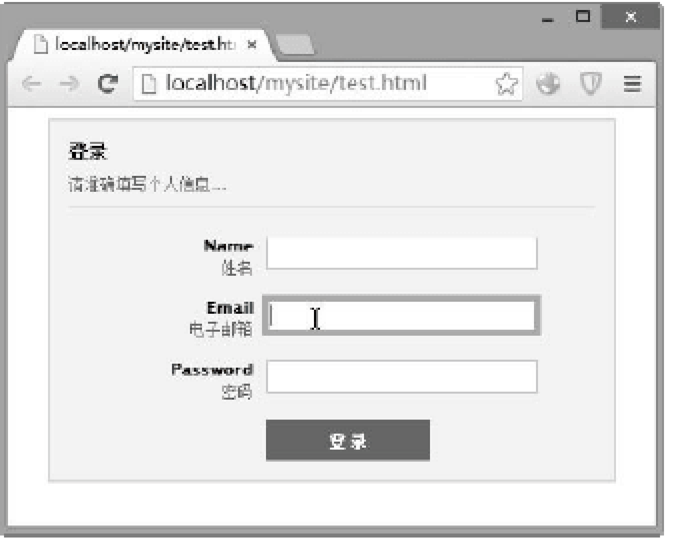
> **注意：** outline 创建的轮廓线是画在一个框“上面”，也就是说，轮廓线总是在顶上，不会影响该框或任何其他框的尺寸。因此，显示或不显示轮廓线不会影响文档流，也不会破坏网页布局。

轮廓线可能是非矩形的。例如，如果元素被分割在好几行，那么轮廓线就至少是能要包含该元素所有框的外廓。和边框不同的是，外廓在线框的起讫端都不是开放的，它总是完全闭合的。

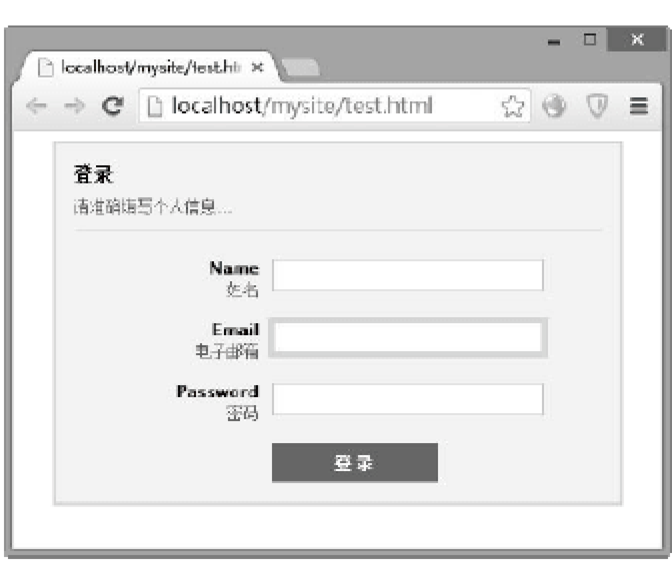
【示例】下面示例设计当文本框获得焦点时，在周围画一个粗实线外廓，提醒用户交互效果，效果如图 22.4 所示。



（a）默认状态



（b）激活状态

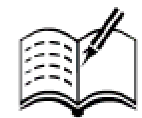


（c）获取焦点状态

图 22.4　设计文本框的轮廓线

```
<style type="text/css">
/*统一页面字体和大小*/
body {
    font-family:"Lucida Grande", "Lucida Sans Unicode", Verdana, Arial, Helvetica, sans-serif;
    font-size:12px;
}
/*清除常用元素的边界、补白、边框默认样式*/
```

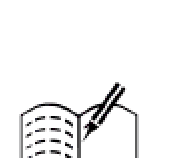




```
p, h1, form, button { border:0; margin:0; padding:0;}
/*定义一个强制换行显示类*/
.spacer { clear:both; height:1px;}
/*定义表单外框样式*/
.myform {margin:0 auto; width:400px; padding:14px;}
/*定制当前表单样式*/
#stylized { border:solid 2px #b7ddf2; background:#ebf4fb;}
/*设计表单内 div 和 p 通用样式效果*/
#stylized h1 {font-size:14px; font-weight:bold;margin-bottom:8px;}
#stylized p {
    font-size:11px; color:#666666;
    margin-bottom:20px; padding-bottom:10px;
    border-bottom:solid 1px #b7ddf2;
}
#stylized label {/*定义表单标签样式*/
    display:block; width:140px;
    font-weight:bold; text-align:right;
    float:left;
}
/*定义小字体样式类*/
#stylized .small {
    color:#666666; font-size:11px; font-weight:normal; text-align:right;
    display:block; width:140px;
}
/*统一输入文本框样式*/
#stylized input {
    float:left;
    font-size:12px;
    padding:4px 2px; margin:2px 0 20px 10px;
    border:solid 1px #aacfe4; width:200px;
}
/*定义图形化按钮样式*/
#stylized button {
    clear:both;
    margin-left:150px;
    width:125px; height:31px;
    background:#666666 url(images/button.png) no-repeat;
    text-align:center; line-height:31px; color:#FFFFFF; font-size:11px; font-weight:bold;
}
/*设计表单内文本框和按钮在被激活和获取焦点状态下时，轮廓线的宽、样式和颜色*/
input:focus, button:focus { outline: thick solid #b7ddf2 }
input:active, button:active   { outline: thick solid #aaa }
</style>
<div id="stylized" class="myform">
    <form id="form1" name="form1" method="post" action="">
        <h1>登录</h1>
        <p>请准确填写个人信息...</p>
        <label>Name <span class="small">姓名</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <label>Email <span class="small">电子邮箱</span> </label>
        <input type="text" name="textfield" id="textfield" />
```



```
            <label>Password <span class="small">密码</span> </label>
            <input type="text" name="textfield" id="textfield" />
            <button   type="submit">登 录</button>
            <div class="spacer"></div>
        </form>
</div>
```



Note



视频讲解

22.2.2 设计轮廓线

CSS3 为轮廓定义了很多属性，使用这些属性可以设计轮廓线样式。

1. 设置宽度

outline-width 属性可以设置轮廓线的宽度。基本语法如下所示：

```
outline-width:<length> | thin | medium | thick
```

取值简单说明如下：

☑ <length> ：定义轮廓粗细的值。

☑ thin：定义细轮廓。

☑ medium：定义中等的轮廓，为默认值。

☑ thick：定义粗的轮廓。

注意： 只有当轮廓样式不是 none 时，该属性才会起作用。如果样式为 none，宽度实际上会重置为 0。不允许设置负长度值。

2. 设置样式

outline-style 属性可以设置轮廓线的样式。基本语法如下所示：

```
outline-style:none | dotted | dashed | solid | double | groove | ridge | inset | outset
```

取值简单说明如下：

☑ none：无轮廓，为默认值。

☑ dotted：点状轮廓。

☑ dashed：虚线轮廓。

☑ solid：实线轮廓。

☑ double：双线轮廓。两条单线与其间隔的和等于指定 outline-width 值。

☑ groove：3D 凹槽轮廓。

☑ ridge：3D 凸槽轮廓。

☑ inset：3D 凹边轮廓。

☑ outset：3D 凸边轮廓。

3. 设置颜色

outline-color 属性可以设置轮廓线的颜色。基本语法如下所示：

```
outline-color:<color> | invert
```

取值简单说明如下：

☑ <color>：指定颜色。

☑ invert：使用背景色的反色。该参数值目前仅在 IE 及 Opera 浏览器下有效。





Note

4. 设置偏移

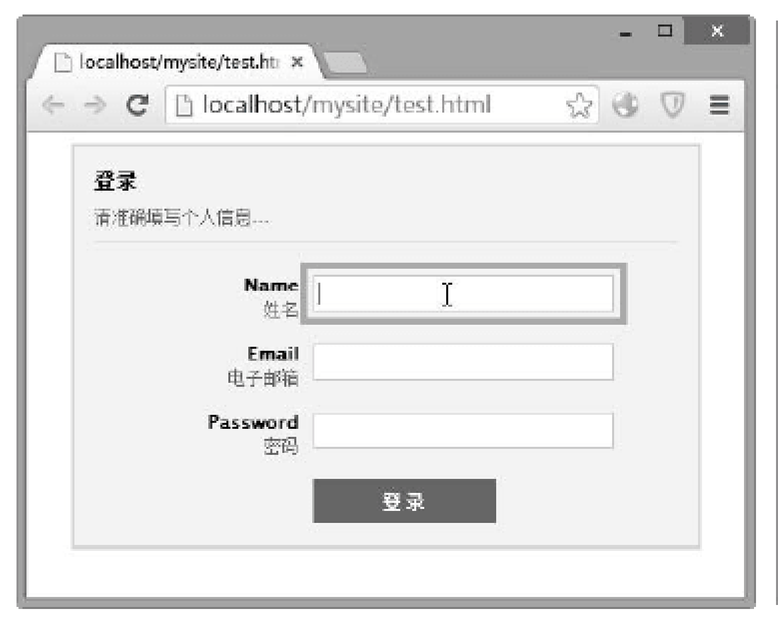
outline-offset 属性可以设置轮廓线的偏移位置。基本语法如下所示：

```
outline-offset:<length>
```

用长度值来定义轮廓偏移，允许负值，默认值为 0。

【示例 1】在 22.2.1 节示例基础上，通过 outline-offset 属性放大轮廓线，使其看起来更大方，演示效果如图 22.5 所示。下面代码仅显示局部 CSS 样式，完整示例样式和结构请参考上节示例代码。

```
<style type="text/css">
……
/*设计表单内文本框和按钮在被激活和获取焦点状态下时，轮廓线的宽、样式和颜色*/
input:focus, button:focus { outline: thick solid #b7ddf2 }
input:active, button:active   { outline: thick solid #aaa }
/*通过 outlineoffset 属性放大轮廓线*/
input:active, button:active { outline-offset: 4px; }
input:focus, button:focus { outline-offset: 4px; }
</style>
<div id="stylized" class="myform">
    <form id="form1" name="form1" method="post" action="">
        <h1>登录</h1>
        ……
    </form>
</div>
```



（a）激活状态



（b）获取焦点状态

图 22.5　放大激活和焦点提示框

【示例 2】下面示例为段落文本中部分文字定义轮廓线，演示效果如图 22.6 所示。

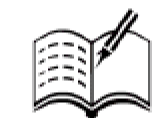


图 22.6　轮廓边框效果

```
<style type="text/css">
.outline { outline: red solid 2px;}
</style>
```



```
<p><b>注释：</b>只有在规定了 !DOCTYPE 时，<span class="outline">Internet Explorer 8 （以及更高版本）</span>才支持 outline 属性。</p>
```



Note

22.3 边框样式

边框是 CSS 盒模型重要组成部分，本节将介绍 CSS3 增强的边框样式，包括图像边框和圆角边框。

权威参考：http://www.w3.org/TR/css3-border/。



视频讲解

22.3.1 定义边框图像源

CSS3 新增的 border-image 属性能够模拟 background-image 属性功能，且功能更加强大，该属性的基本语法如下所示：

```
border-image:<' border-image-source '> || <' border-image-slice '> [ / <' border-image-width '> | / <' border-image-width '>? / <' border-image-outset '> ]? || <' border-image-repeat '>
```

取值说明如下：

☑ <' border-image-source '>：设置对象的边框是否用图像定义样式或图像来源路径。

☑ <' border-image-slice '>：设置边框图像的分割方式。

☑ <' border-image-width '>：设置对象的边框图像宽度。

☑ <' border-image-outset '>：设置对象的边框图像的扩展。

☑ <' border-image-repeat '>：设置对象的边框图像的平铺方式。

【示例】下面示例为元素 div 定义边框图像，使用 border-image-source 导入外部图像源 images/border1.png，根据 border-image-slice 属性，值为(27 27 27 27)，把图像切分为 9 块，然后分别把这九块图像切片按顺序填充到边框四边、四角和内容区域。示例主要代码如下：

```
<style type="text/css">
div {
    height:160px;
    border:solid 27px;
    /*设置边框图像*/
    border-image: url(images/border1.png) 27;
</style>
<div></div>
```

页面浏览效果如图 22.7 所示。

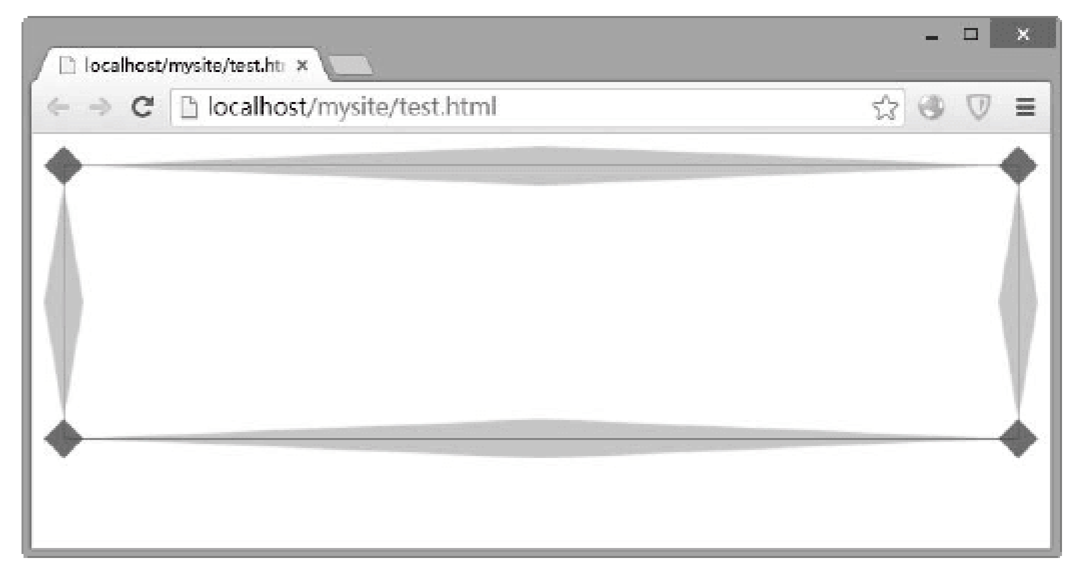


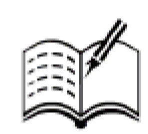
图 22.7　定义边框背景样式



在上面示例中，使用了一个 71px×71px 大小的图像，在这个正方形的图像中，被等分了 9 个方块，每个方块的高和宽都是 21px×21px 大小。当声明 border-image-slice 属性值为(27 27 27 27)时，则按下面说明进行解析：

- ☑ 第一个参数值表示从上向下裁切图像，显示在顶边。
- ☑ 第二个参数值表示从右向左裁切图像，显示在右边。
- ☑ 第三个参数值表示从下向上裁切图像，显示在底边。
- ☑ 第四个参数值表示从左向右裁切图像，显示在左边。

图像被四个参数值裁切为 9 块，再根据边框的大小进行自适应显示。例如，当分别设置边框为不同大小，则显示效果除了粗细之外，其他则都是完全相同的。



Note



视 频 讲 解

22.3.2 定义边框图像平铺方式

border-image-repeat 属性设置对象的边框图像的平铺方式。该属性的基本语法如下所示：

```
border-image-repeat:[ stretch | repeat | round | space ]{1,2}
```

取值简单说明如下。

- ☑ stretch：用拉伸方式来填充边框图像，为默认值。
- ☑ repeat：用平铺方式来填充边框图像。当图片碰到边界时，如果超过则被截断。
- ☑ round：用平铺方式来填充边框图像。图像会根据边框的尺寸动态调整图像的大小直至正好可以铺满整个边框。
- ☑ space：用平铺方式来填充边框图像。图像会根据边框的尺寸动态调整图像之间的距离直至正好可以铺满整个边框。

【示例】下面示例以 22.3.1 节示例为基础，设置边框图像平铺显示："border-image-repeat:round;"，演示效果如图 22.8 所示。

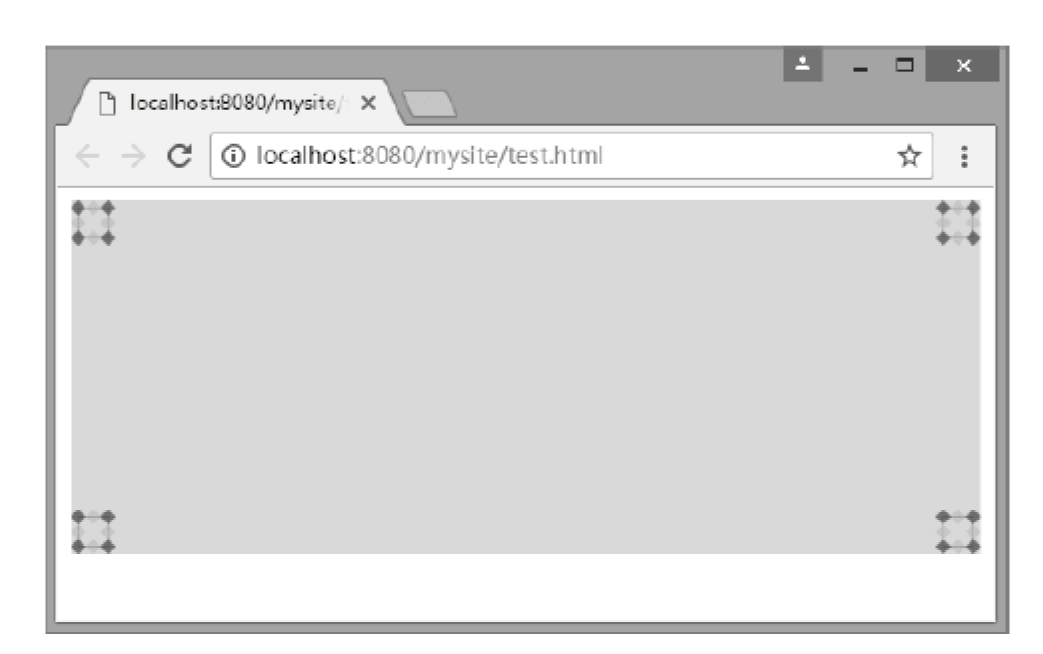
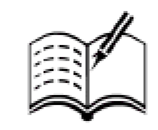


图 22.8 定义边框图像平铺显示

```
<style type="text/css">
div {
    height:160px;
    background:hsla(93,96%,62%,1.00);
    border:solid 27px red;
    /*设置边框图像源*/
    border-image-source: url(images/border1.png);
    /*设置边框图像的平铺方式*/
    border-image-repeat:round;
```

```
}
</style>
```

22.3.3　定义边框图像宽度




border-image-width 属性设置对象的边框图像的宽度。该属性的基本语法如下所示：

```
border-image-width:[ <length> | <percentage> | <number> | auto ]{1,4}
```

取值简单说明如下。

☑ <length>：用长度值指定宽度。不允许负值。

☑ <percentage>：用百分比指定宽度。参照其包含块进行计算，不允许负值。

☑ <number>：用浮点数指定宽度。不允许负值。

☑ auto：如果 auto 值被设置，则<' border-image-width '>采用与<' border-image-slice '>相同的值。

【示例】下面示例以 22.3.1 节示例为基础，设置边框背景平铺显示："border-image-repeat:round;"，图像宽度为 500px，演示效果如图 22.9 所示。

```
<style type="text/css">
div {
    height:160px;
    background:hsla(93,96%,62%,1.00);
    border:solid 27px red;
    /*设置边框图像源*/
    border-image-source: url(images/border1.png);
    /*设置边框图像的平铺方式*/
    border-image-repeat:round;
    /*设置边框图像的宽度*/
    border-image-width: 500px;
}
</style>
<div>border-image-source: url(images/border1.png);<br>
    border-image-repeat:round;<br>
    border-image-width:500px;</div>
```



图 22.9　定义边框图像宽度

22.3.4　定义边框图像分割方式

border-image-slice 属性设置对象的边框图像的分割方式。该属性的基本语法如下所示：



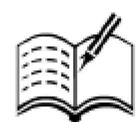
```
border-image-slice:[ <number> | <percentage> ]{1,4} && fill?
```

取值简单说明如下。

☑ <number>：用浮点数指定宽度。不允许负值。

☑ <percentage>：用百分比指定宽度。参照其包含块区域进行计算，不允许负值。

☑ fill：保留裁减后的中间区域，其铺排方式遵循<' border-image-repeat '>的设定。



【示例】下面示例以 22.3.1 节示例为基础，设置边框背景平铺显示："border-image-repeat:round;"，设置裁切值为 10："border-image-slice: 10;"，演示效果如图 22.10 所示。

```
<style type="text/css">
div {
    height:160px;
    background:hsla(93,96%,62%,1.00);
    border:solid 27px red;
    /*设置边框图像源*/
    border-image-source: url(images/border1.png);
    /*设置边框图像的平铺方式*/
    border-image-repeat:round;
    /*设置边框图像的宽度*/
    border-image-width: 500px;
    /*设置边框图像的裁切值为 10*/
    border-image-slice: 10;
}
</style>
<div>border-image-source: url(images/border1.png);<br>
    border-image-repeat:round;<br>
    border-image-slice: 10;</div>
```

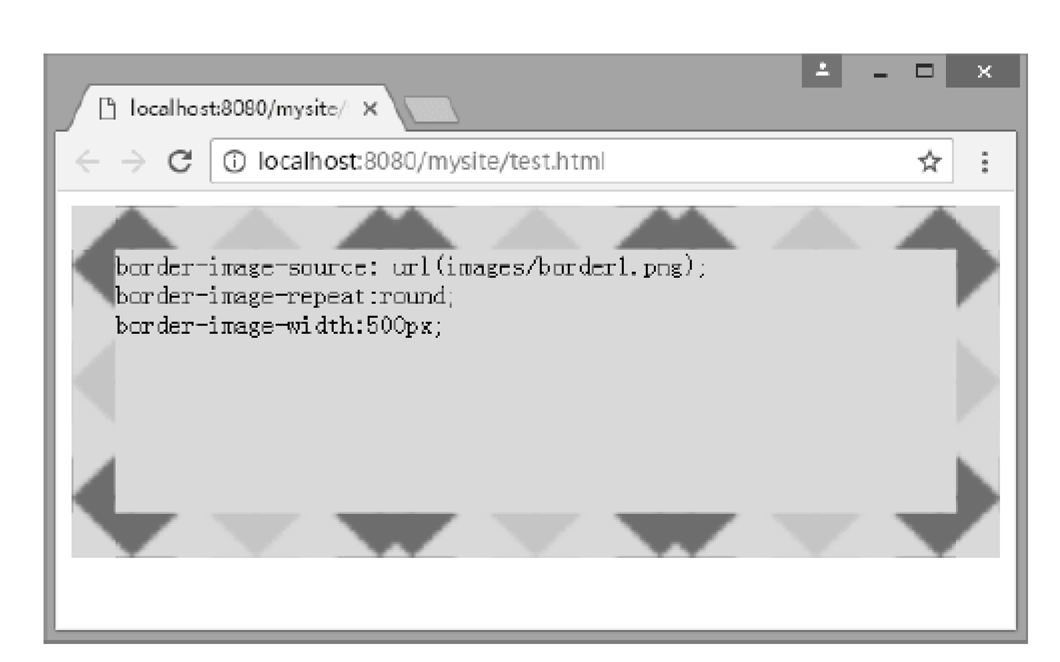


图 22.10　定义边框图像裁切值

22.3.5　定义边框图像扩展



border-image-outset 属性设置对象的边框图像的扩展。该属性的基本语法如下所示：

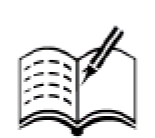
```
border-image-outset:[ <length> | <number> ]{1,4}
```

取值简单说明如下。

☑ <length>：用长度值指定宽度。不允许负值。

☑ <number>：用浮点数指定宽度。不允许负值。





Note

【示例】下面以 22.3.1 节示例为基础，设置边框背景向外扩展 50px，演示效果如图 22.11 所示。

```
<style type="text/css">
div {
    height:160px;
    margin:60px;
    background:hsla(93,96%,62%,1.00);
    border:solid 27px red;
    /*设置边框图像源*/
    border-image-source: url(images/border1.png);
    /*设置边框图像的平铺方式*/
    border-image-repeat:round;
    /*设置边框图像的宽度*/
    border-image-width: 500px;
    /*设置边框图像的裁切值为 10*/
    border-image-slice: 10;
    /*设置边框图像向外扩展 50 像素*/
    border-image-outset: 50px;
}
</style>
<div>border-image-source: url(images/border1.png);<br>
    border-image-repeat:round;<br>
    border-image-slice: 10;<br>
    border-image-outset: 50px;</div>
```

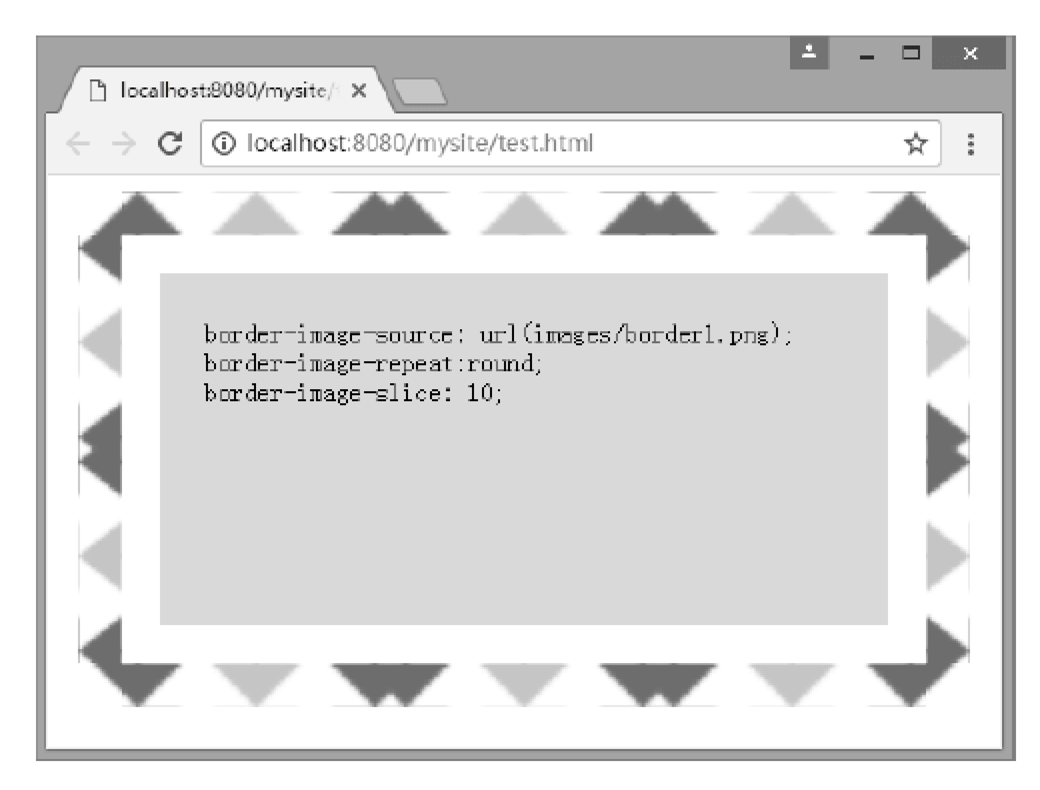


图 22.11　定义边框图像向外扩展



视频讲解

22.3.6　案例：应用边框图像

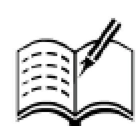
下面结合示例介绍 border-image 在页面中的应用。

【示例 1】下面示例演示如何设计左下和右下圆角显示，演示效果如图 22.12 所示。

```
<style type="text/css">
div {
    height: 120px;
    text-align:center;
    border-style:solid;
    border-width: 10px;
```



```
        border-image: url(images/r2.png) 20;
    }
    </style>
    <div>border-image: url(images/r2.png) 20; 图像源效果如下所示：<br>
        <img src="images/r2.png" /></div>
```



【示例 2】设计完全圆角边框效果。设计圆角图像大小为 42px×42px，裁切半径为 20px，显示效果如图 22.13 所示。

```
    <style type="text/css">
    div {
        height: 120px;
        text-align:center;
        border-style:solid;
        border-width: 10px;
        border-image: url(images/r3.png) 20;
    }
    </style>
    <div>border-image: url(images/r3.png) 20; 图像源效果如下所示：<br>
        <img src="images/r3.png" /></div>
```





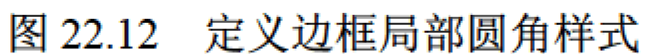


图 22.12　定义边框局部圆角样式　　　图 22.13　定义完全圆角边框样式

【示例 3】设计阴影特效。设计边框图像大小为 42px×42px，显示效果如图 22.14 所示。

```
    <style type="text/css">
    img {
        height:400px;
        border-style:solid;
        border-width:2px 5px 6px 2px;
        border-image: url(images/r4.png)   2 5 6 2;
    }
    </style>

    <img src="images/2.jpg" />
```

【示例 4】设计选项卡。设计边框图像大小为 12px×27px，圆角半径为 12px，显示效果如图 22.15 所示。

```
    <style type="text/css">
    ul{
        margin:0; padding:0;
        list-style-type:none;
    }
    li {
```



```
        width:100px; height:20px;
        border-style:solid;
        cursor:pointer;
        float:left;
        padding:4px 0;
        text-align:center;
        border-width:5px 5px 0px;
        border-image: url(images/r5.png) 5 5 0;
    }
    </style>
    <ul>
        <li>首页</li>
        <li>咨询</li>
        <li>关于</li>
    </ul>
```

Note

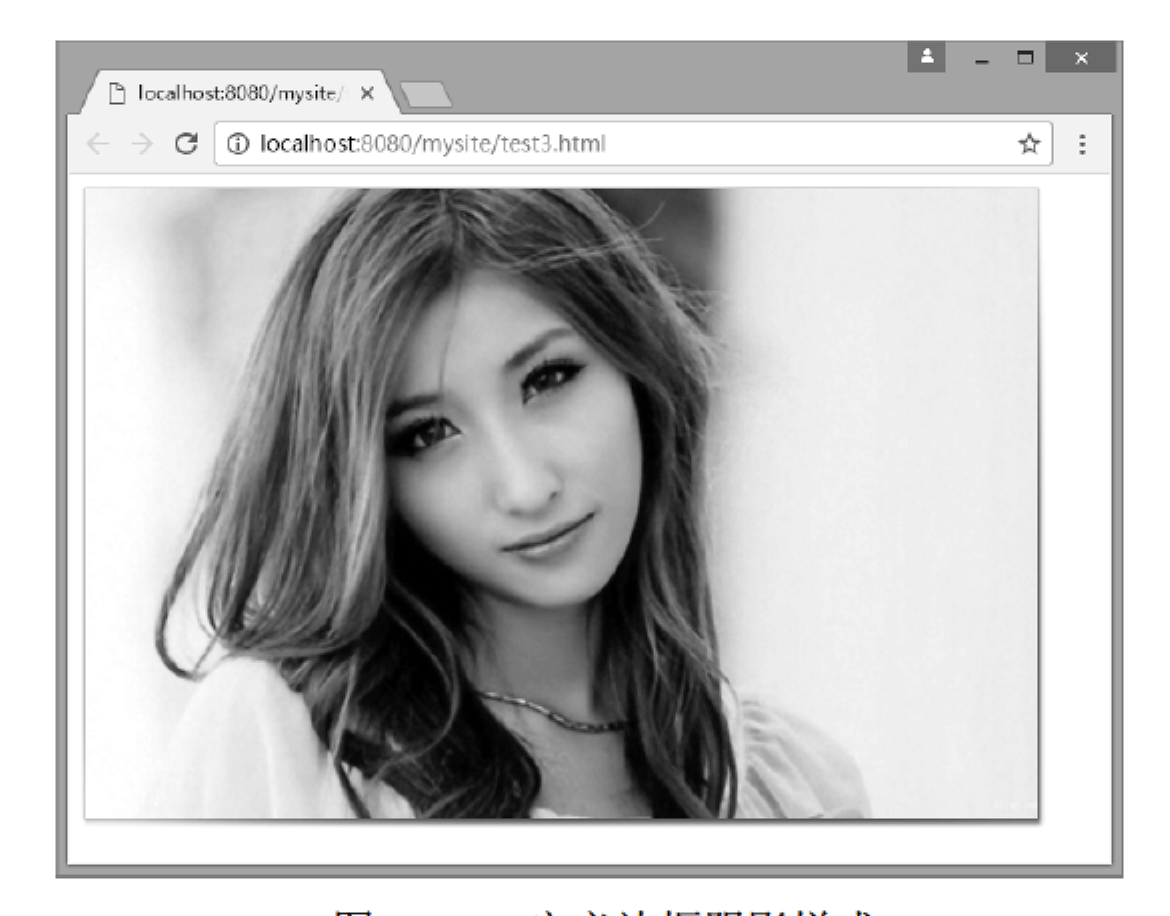


图 22.14　定义边框阴影样式

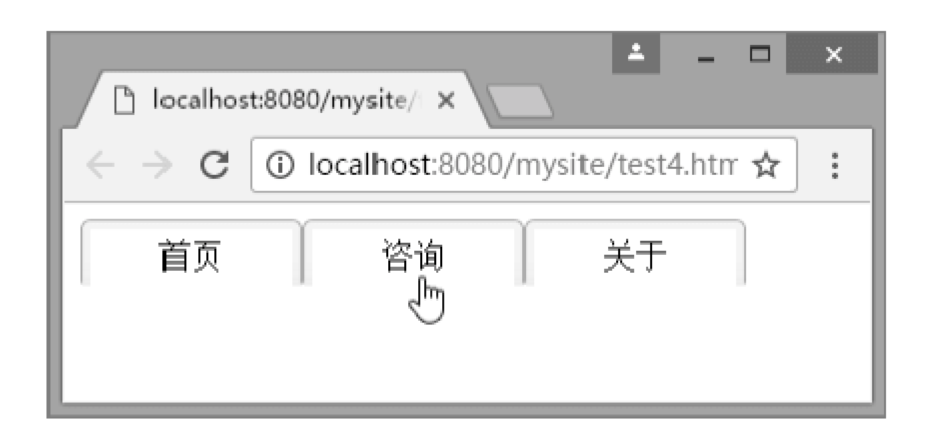


图 22.15　定义选项卡样式



视频讲解

22.3.7　定义圆角边框

CSS3 新增 border-radius 属性，使用它可以设计元素的边框以圆角样式显示。border-radius 属性的基本语法如下所示：

```
border-radius:[ <length> | <percentage> ]{1,4} [ / [ <length> | <percentage> ]{1,4} ]?
```

取值简单说明如下：

☑ <length>：用长度值设置对象的圆角半径长度。不允许负值。

☑ <percentage>：用百分比设置对象的圆角半径长度。不允许负值。

为了方便定义四个顶角的圆角，border-radius 属性派生了 4 个子属性。

☑ border-top-right-radius：定义右上角的圆角。

☑ border-bottom-right-radius：定义右下角的圆角。

☑ border-bottom-left-radius：定义左下角的圆角。

☑ border-top-left-radius：定义左上角的圆角。





Note

提示： border-radius 属性可包含两个参数值：第一个值表示圆角的水平半径，第二个值表示圆角的垂直半径，两个参数值通过斜线分隔。如果仅包含一个参数值，则第二个值与第一个值相同，它表示这个角就是一个四分之一圆角。如果参数值中包含 0，则这个角就是矩形，不会显示为圆角。

针对 border-radius 属性参数值，各种浏览器的处理方式并不一致。在 Chrome 和 Safari 浏览器中，会绘制出一个椭圆形边框，第一个半径为椭圆的水平方向半径，第二个半径为椭圆的垂直方向半径。在 Firefox 和 Opera 浏览器中，将第一个半径作为边框左上角与右下角的圆半径来绘制，将第二个半径作为边框右上角与左下角的圆半径来绘制。

【示例 1】 下面给 border-radius 属性设置一个值："border-radius:10px;"，演示效果如图 22.16 所示。

```
<style type="text/css">
img {
    height:300px;
    border:1px solid red;
    border-radius:10px;
}
</style>
<img src="images/1.jpg" />
```

如果为 border-radius 属性设置两个参数，则效果如图 22.17 所示。

```
img {
    height:300px;
    border:1px solid red;
    border-radius:20px/40px;
}
```

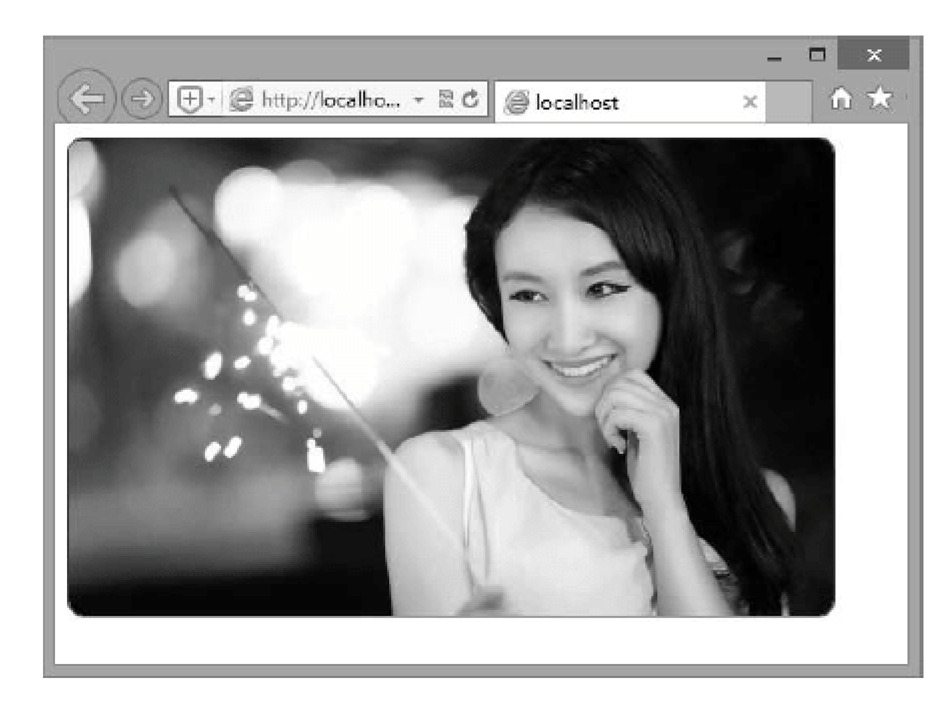




图 22.16　定义圆角样式 1

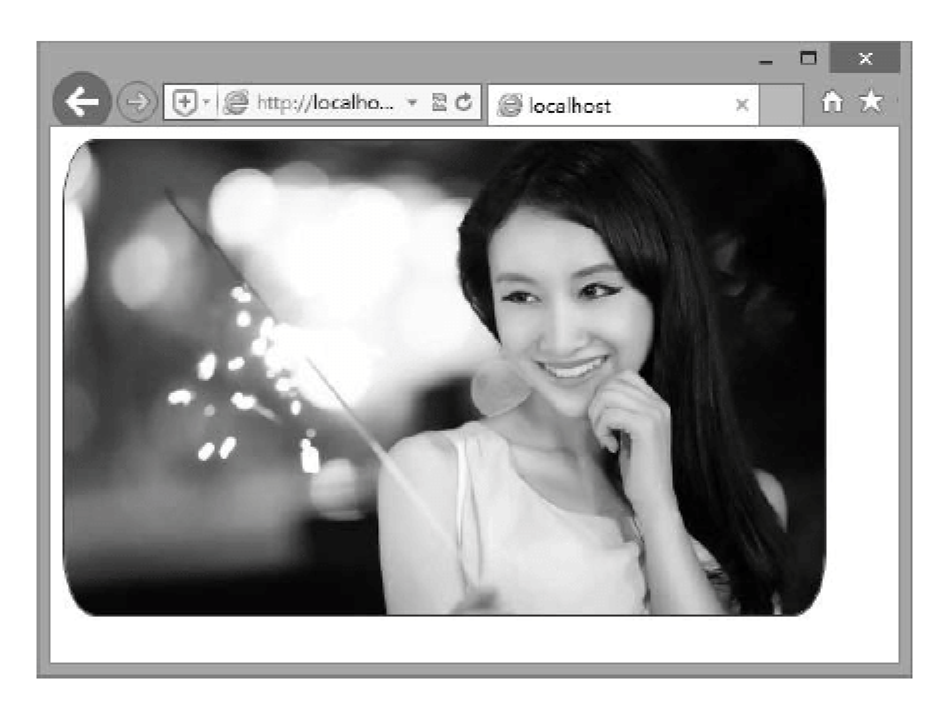


图 22.17　定义圆角样式 2

也可以为元素的四个顶角定义不同的值，实现的方法有两种：

一种方法是利用 border-radius 属性，为其赋一组值。当为 border-radius 属性赋一组值，将遵循 CSS 赋值规则，可以包含 2 个、3 个或者 4 个值集合。但是此时无法使用斜杠方式定义圆角水平和垂直半径。

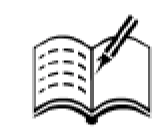
如果是四个值，则这四个值将按照 top-left、top-right、bottom-right、bottom-left 顺序来设置。

如果 bottom-left 值省略，那么它等于 top-right。

如果 bottom-right 值省略，那么它等于 top-left。

如果 top-right 值省略，那么它等于 top-left。





Note

如果为 border-radius 属性设置 4 个值的集合参数，则每个值表示每个角的圆角半径。

【示例 2】下面示例为图像的四个顶角定义不同圆角半径，演示效果如图 22.18 所示。

```
img {
    height:300px;
    border:1px solid red;
    border-radius:10px 30px 50px 70px;
}
```

如果为 border-radius 属性设置 3 个值的集合参数，则第一个值表示左上角的圆角半径，第二个值表示右上和左下两个角的圆角半径，第三个值表示右下角的圆角半径。

如果为 border-radius 属性设置 2 个值的集合参数，则第一个值表示左上角和右下角的圆角半径，第二个值表示右上和左下两个角的圆角半径。

另一种方法是利用派生子属性进行定义，如 border-top-right-radius、border-bottom-right-radius、border-bottom-left-radius、border-top-left-radius。

注意：Gecko 和 Presto 引擎在写法上存在很大差异。

【示例 3】下面代码定义 div 元素右上角为 50 像素的圆角，演示效果如图 22.19 所示。

```
img {
    height:300px;
    border:1px solid red;
    -moz-border-radius-topright:50px;
    -webkit-border-top-right-radius:50px;
    border-top-right-radius:50px;
}
```



图 22.18　分别定义不同顶角的圆角样式

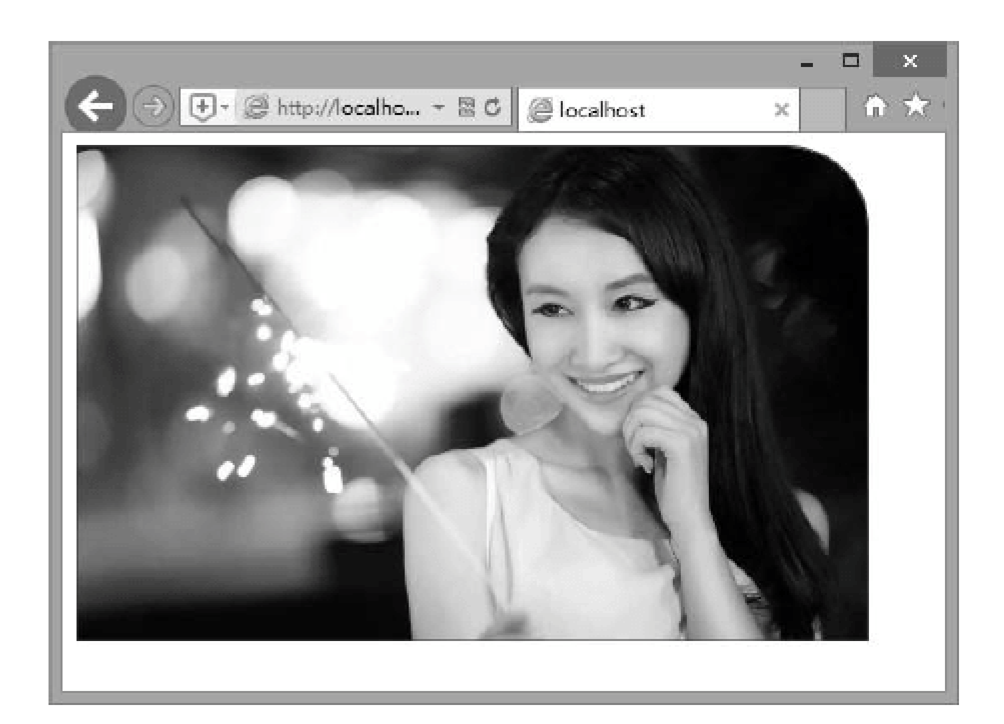


图 22.19　定义某个顶角的圆角样式

22.3.8　案例：设计椭圆图形



视 频 讲 解

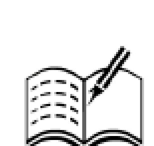
使用 border-radius 属性设计圆角时，可能会存在下面几种情况：

☑ 如果受影响的角的两个相邻边宽度不同，那么这个圆角将会从宽的一边圆滑过渡到窄的一边，即偏向宽边的圆弧略大，而偏向窄边的圆弧略小。

☑ 如果两条边宽度相同，那么圆角两个相邻边呈对称圆弧显示，即相交 45° 的对称线上。

☑ 如果一条边宽度是相邻另一条边宽度的两倍，那么两边圆弧线交于靠近窄边的 30° 角线上。





border-radius 不允许圆角彼此重叠，当相邻两个圆角的半径之和大于元素的宽或高时，在浏览器中会呈现为椭圆或正圆效果。

【示例】下面代码定义 img 元素显示为圆形，当图像宽高比不同时，显示效果不同，如图 22.20 所示。

```
<style type="text/css">
img {/*定义图像圆角边框*/
    border: solid 1px red;
    border-radius: 50%; /*圆角*/
}
.r1 {/*定义第 1 幅图像宽高比为 1：1*/
    width:300px;
    height:300px;
}
.r2 {/*定义第 2 幅图像宽高比不为 1：1*/
    width:300px;
    height:200px;
}
.r3 {/*定义第 3 幅图像宽高比不为 1：1*/
    width:300px;
    height:100px;
    border-radius: 20px; /*定义圆角*/
}
</style>
<img class="r1" src="images/1.jpg" title="圆角图像" />
<img class="r2" src="images/1.jpg" title="椭圆图像" />
<img class="r3" src="images/1.jpg" title="圆形图像" />
```



图 22.20　定义圆形显示的元素效果

22.4　盒子阴影

CSS3 的 box-shadow 类似于 text-shadow，不过 text-shadow 负责为文本设置阴影，而 box-shadow 负责给对象定义图层阴影效果。

权威参考：http://www.w3.org/TR/css3-border/。



权威参考



视频讲解

Note

22.4.1 定义盒子阴影

box-shadow 属性可以定义元素的阴影，基本语法如下所示：

```
box-shadow : none | inset? && <length>{2,4} && <color>?
```

取值简单说明如下：

☑ none：无阴影。
☑ inset：设置对象的阴影类型为内阴影。该值为空时，对象的阴影类型为外阴影。
☑ <length>①：第 1 个长度值用来设置对象的阴影水平偏移值。可以为负值。
☑ <length>②：第 2 个长度值用来设置对象的阴影垂直偏移值。可以为负值。
☑ <length>③：如果提供了第 3 个长度值，则用来设置对象的阴影模糊值。不允许负值。
☑ <length>④：如果提供了第 4 个长度值，则用来设置对象的阴影外延值。可以为负值。
☑ <color>：设置对象的阴影的颜色。

下面结合案例进行演示说明。

【示例 1】下面示例定义一个简单的实影投影效果，演示效果如图 22.21 所示。

```
<style type="text/css">
img{
    height:300px;
    box-shadow:5px 5px;
}
</style>
<img src="images/1.jpg" />
```

【示例 2】定义位移、阴影大小和阴影颜色，则演示效果如图 22.22 所示。

```
img{
    height:300px;
    box-shadow:2px 2px 10px #06C;
}
```



图 22.21　定义简单的阴影效果

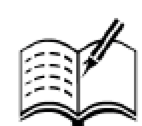


图 22.22　定义复杂的阴影效果

【示例 3】定义内阴影，阴影大小为 10px，颜色为#06C，则演示效果如图 22.23 所示。

```
<style type="text/css">
pre {
```





Note

```
        padding: 26px;
        font-size:24px;
        box-shadow: inset 2px 2px 10px #06C;
}
</style>
<pre>
-moz-box-shadow: inset 2px 2px 10px #06C;
-webkit-box-shadow: inset 2px 2px 10px #06C;
box-shadow: inset 2px 2px 10px #06C;
</pre>
```

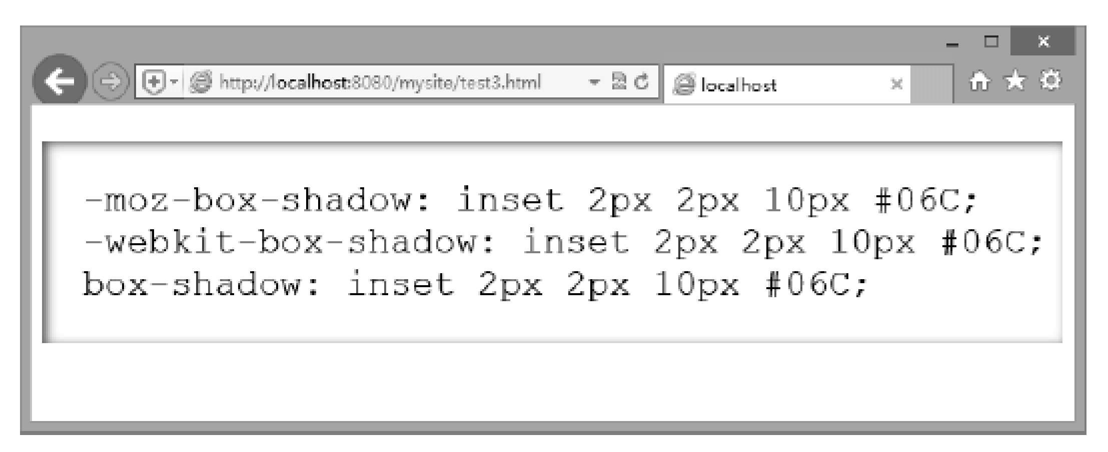


图 22.23　定义内阴影效果

【示例 4】通过设置多组参数值定义多色阴影，演示效果如图 22.24 所示。

```
img {
        height: 300px;
        box-shadow: -10px 0 12px red,
                    10px 0 12px blue,
                    0 -10px 12px yellow,
                    0 10px 12px green;
}
```

【示例 5】通过多组参数值还可以定义渐变阴影，演示效果如图 22.25 所示。





图 22.24　定义多色阴影效果

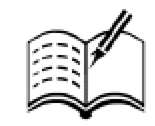


图 22.25　定义渐变阴影效果

注意：当给同一个元素设计多个阴影时，最先写的阴影将显示在最顶层。

```
<!doctype html>
img{
        height:300px;
```

```
        box-shadow:0 0 10px red,
                    2px 2px 10px 10px yellow,
                    4px 4px 12px 12px green;
    }
```


Note

视频讲解

22.4.2 案例：box-shadow 的应用

本节通过一个简单的示例进一步练习 box-shadow 的应用。

【操作步骤】

第 1 步，设计一个简单的盒子，并定义基本形状。

```
<style type="text/css">
.box{
    width:100px; height:100px;                    /*固定大小*/
    text-align:center; line-height:100px;         /*显示在中央*/
    background-color:rgba(255,204,0,.5);          /*浅色背景*/
    border-radius:10px;                           /*适当圆角*/
    padding:10px; margin:10px;                    /*添加间距*/
}
</style>
<div class="box bs1">box-shadow</div>
```

第 2 步，阴影就是对原对象的复制，包括内边距和边框都属于 box 的占位范围，阴影也包括对内边距和边框的复制，但是阴影本身不占据布局的空间。比较效果如图 22.26 所示。

```
.bs1{box-shadow:120px 0px #ccc;}
```

第 3 步，四周有一样模糊值的阴影，如图 22.27 所示。

```
.bs1{ box-shadow:0 0 20px #666;}
```



图 22.26　比较对象和阴影大小



图 22.27　四周同时显示阴影

第 4 步，定义 5px 扩展阴影，如图 22.28 所示。

```
.bs1{ box-shadow:0 0 0 5px #333;}
```

阴影不像 border 要占据布局的空间，因此要实现对象鼠标经过产生外围的边框，可以使用阴影的扩展来代替 border。或者使用 border 的 transparent 实现，不过不如 box-shadow 的 spread 扩展方便。如果使用 border，布局会产生影响。

第 5 步，拓展为负值的阴影，如图 22.29 所示。

```
.bs1 {
```



```
        box-shadow: 0 15px 10px -15px #333;
        border: none;
    }
```

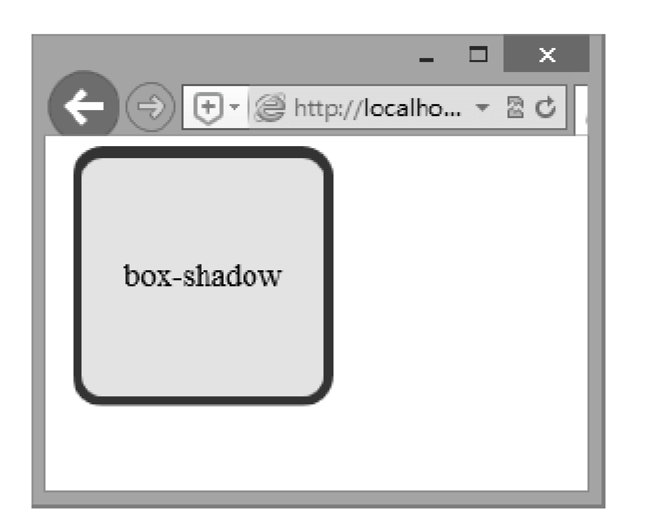




图 22.28　定义扩展阴影

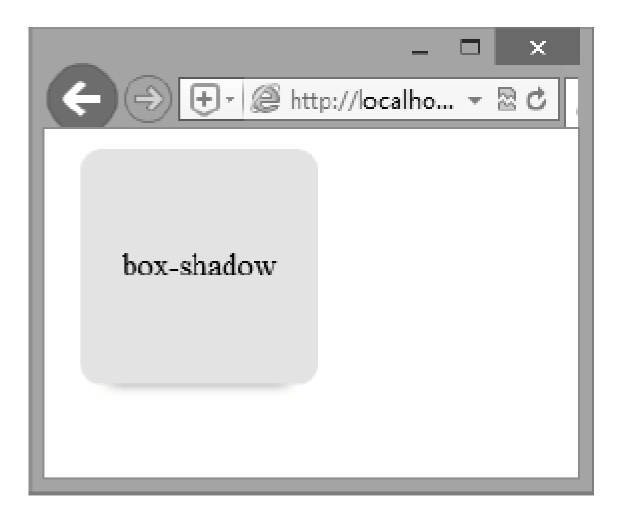
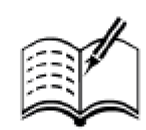


图 22.29　定义负值阴影



注意：要产生图 22.29 的效果，y 轴的值和 spread 的值刚好相反。其他边设计同理。

第 6 步，定义内阴影，如图 22.30 所示。

```
.bs1 {
    background-color: #1C8426;
    box-shadow: 0px 0px 20px #fff inset;
}
```

注意：可以直接为 div 这样的盒子设置 box-shadow 盒阴影，但是不能直接为 img 图片设置盒阴影。

```
/* 直接在图片上添加内阴影，无效*/
.img-shadow img {
    box-shadow: inset 0 0 20px red;
}
```

可以通过为 img 的容器 div 设置内阴影，然后让 img 的 z-index 为-1 解决这个问题。但是这种做法不可以为容器设置背景色，因为容器的级别比图片高，设置了背景色会挡住图片，效果如图 22.31 所示。

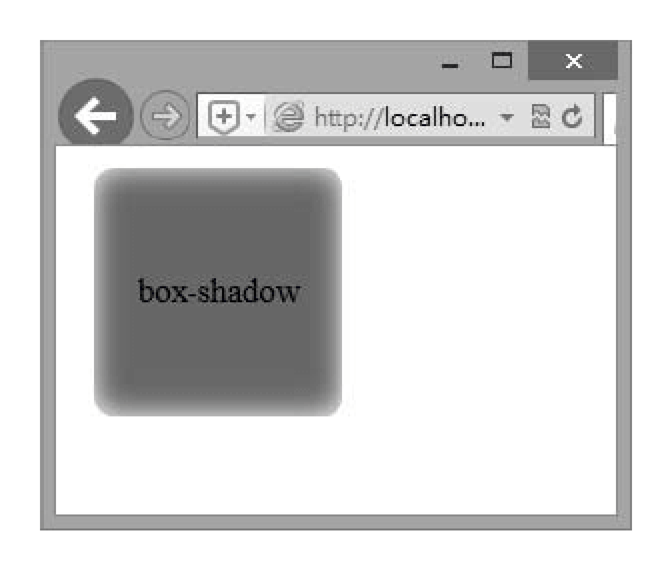


图 22.30　定义内阴影

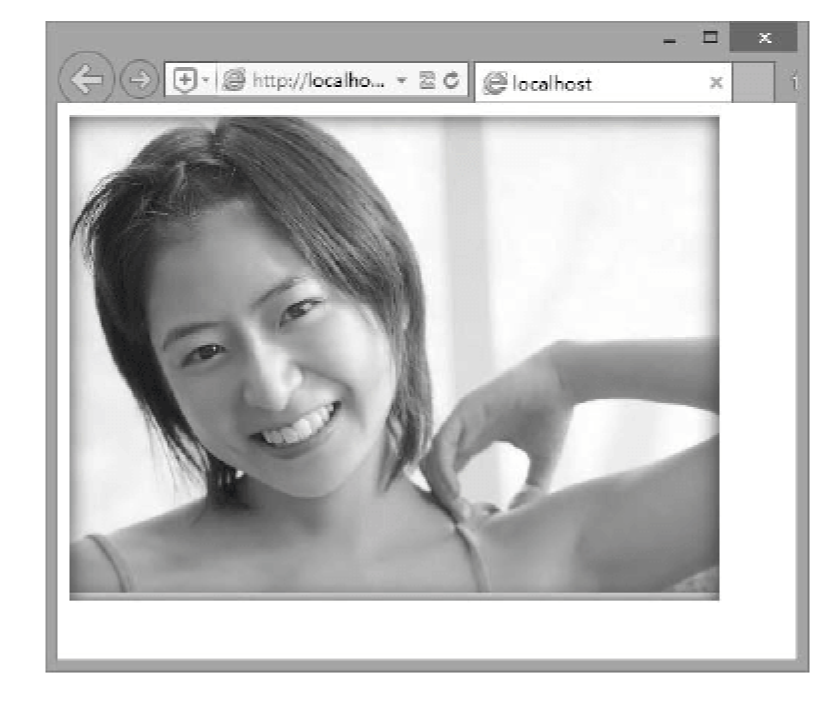
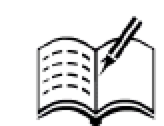


图 22.31　为图片定义内阴影

```
/* 在图片容器上添加内阴影，生效*/
.box-shadow {
    box-shadow: inset 0 0 20px red;
    display:inline-block;
```



```
}
.box-shadow img {
      position: relative;
      z-index: -1;
}
```


Note

第 7 步，还有一个更好的方法，不用考虑图片的层级，利用“:before”伪元素可以实现，而且还可以为父容器添加背景色等。

```
/*给图片容器上添加伪元素或伪类，不用为 img 设置负值的 z-index 值了。有内阴影*/
img {
      position: relative;
      background-color: #FC3;
      padding: 5px;
}
img:before {
      content: '';
      position: absolute;
      top: 0; right: 0; bottom: 0;    left: 0;
      box-shadow: inset 0 0 40px #f00;
}
```

第 8 步，定义多个阴影，如图 22.32 所示。

```
.bs1 {
      box-shadow: 40px 40px rgba(26,202,221,0.5),
      80px 80px rgba(236,43,120,.5);
      border-radius: 0;
}
```

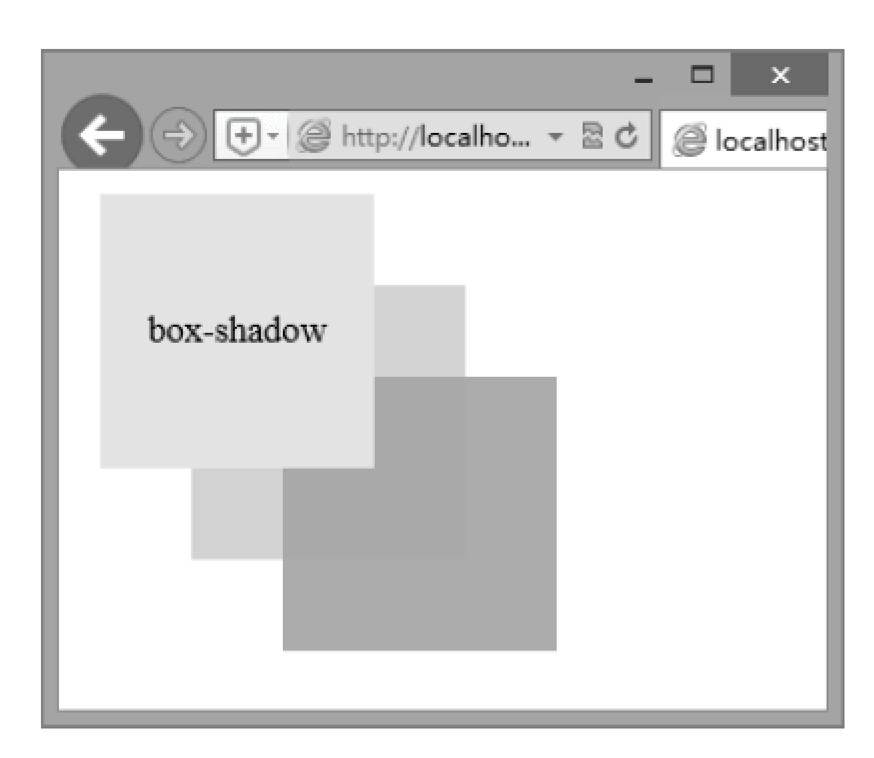


图 22.32　定义多个阴影

提示：阴影也是有层叠关系的，前面的阴影层级高，会压住后面的阴影。阴影和阴影之间的透明度可见，而主体对象的透明度对阴影不起作用。

22.4.3　案例：设计翘边阴影


视频讲解

本例使用 box-shadow 设计翘边阴影，翘边效果就是四角旁边翘起阴影，如图 22.33 所示。



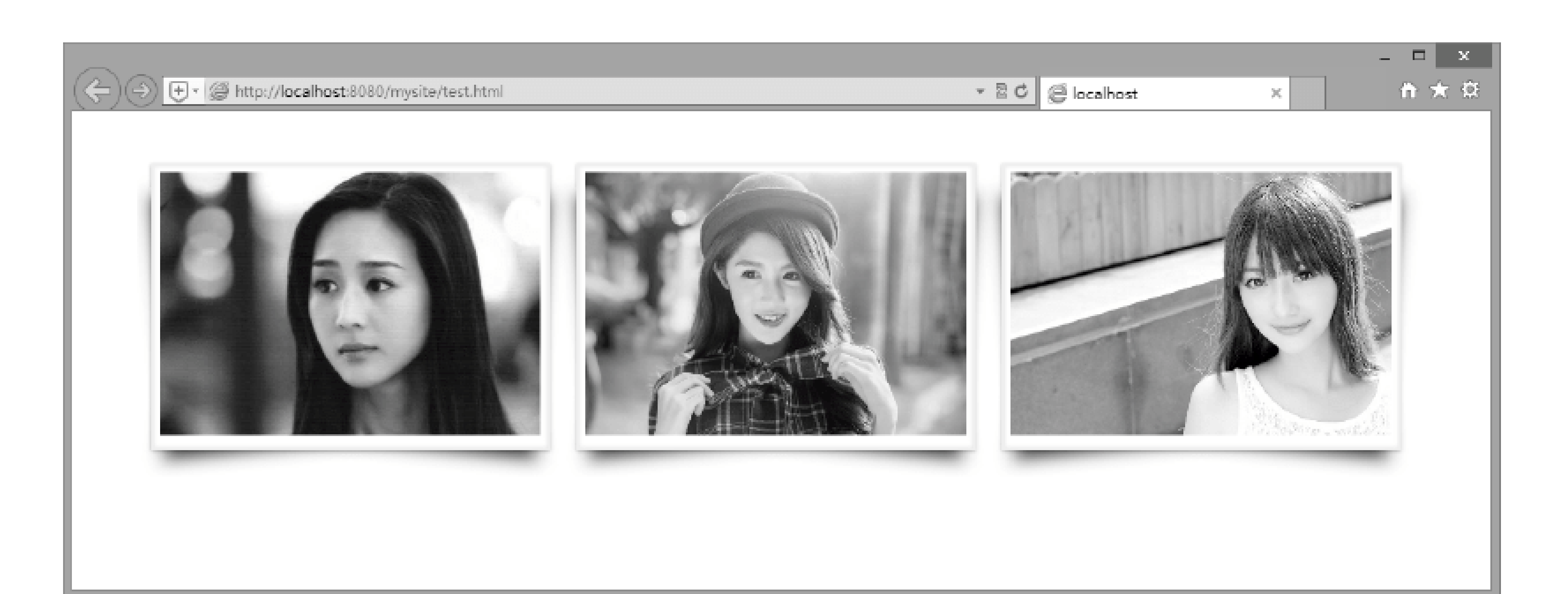
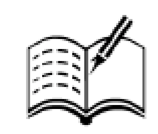


图 22.33　设计翘边阴影效果



Note



示例效果

示例主要代码如下：

```
<style type="text/css">
* { margin: 0; padding: 0;}                    /*清除页边距*/
ul { list-style: none; }                        /*清除项目列表符号*/
.box {                                          /*设计盒子样式*/
    width: 980px; height: auto;                 /*固定大小，高度自动调整*/
    clear: both;
    overflow: hidden;                           /*禁止超出显示*/
    margin: 20px auto;                          /*居中显示*/
}
.box li {                                       /*设计每个图片外框样式*/
    background: #fff;                           /*白色背景*/
    float: left;                                /*浮动并列显示*/
    position: relative;                         /*定义定位包含框*/
    margin: 20px 10px;                          /*调整项目间距*/
    border: 2px solid #efefef;                  /*增加浅色边框*/
    /*添加内阴影*/
    box-shadow: 0 1px 4px rgba(0,0,0,0.27), 0 0 4px rgba(0,0,0,0.1) inset;
}
.box li img {                                   /*固定图片大小，增加外边距*/
    width: 290px; height: 200px;
    margin: 5px;
}
.box li:before {                                /*在左侧添加翘起阴影*/
    content: "";                                /*空内容*/
    position: absolute;                         /*固定定位*/
    width: 90%; height: 80%;                    /*定义大小*/
    bottom: 13px; left: 21px;                   /*定位*/
    background: transparent;                    /*透明背景*/
    z-index: -2;                                /*显示在照片下面*/
    box-shadow: 0 8px 20px rgba(0,0,0,0.8);     /*添加阴影*/
    transform: skew(-12deg) rotate(-6deg);      /*变形并旋转阴影，让其翘起*/
}
.box li:after {                                 /*在右侧添加翘起阴影，方法同上*/
    content: "";
    position: absolute;
    width: 90%;height: 80%;
```

```
        bottom: 13px; right: 21px;
        z-index: -2;
        background: transparent; box-shadow: 0 8px 20px rgba(0,0,0,0.8);
        transform: skew(12deg) rotate(6deg);
    }
    </style>
    <ul class="box">
        <li><img src="images/1.jpg" /></li>
        <li><img src="images/2.jpg" /></li>
        <li><img src="images/3.jpg" /></li>
    </ul>
```



Note

本例主要使用 CSS3 的伪类“:before”和“:after”，分别在被插盒子里面的内容前面和内容后面动态插入空内容。设置盒子时，每个大盒子小盒子的值的大小都要算清楚，不要超过大盒子范围，而且也不要浪费。使用 z-index 属性设置元素的堆叠顺序。拥有更高堆叠顺序的元素总是会处于堆叠顺序较低的元素的前面，它仅能在定位元素上奏效。

skew()函数能够让元素倾斜显示，它可以将一个对象以其中心位置围绕着 X 轴和 Y 轴按照一定的角度倾斜。rotate()函数只是旋转，而不会改变元素的形状。skew()函数不会旋转，而只会改变元素的形状。相关知识将在后面章节介绍。

22.5 案例实战

本节将通过两个案例练习 CSS 盒模型相关组成要素的具体应用。



视频讲解

22.5.1 设计内容页

本例将应用 box-shadow、text-shadow 和 border-radius 等属性，定义一个包含阴影、圆角特效，同时利用 CSS 渐变、半透明特效设计的精致栏目效果，预览效果如图 22.34 所示。



线上阅读

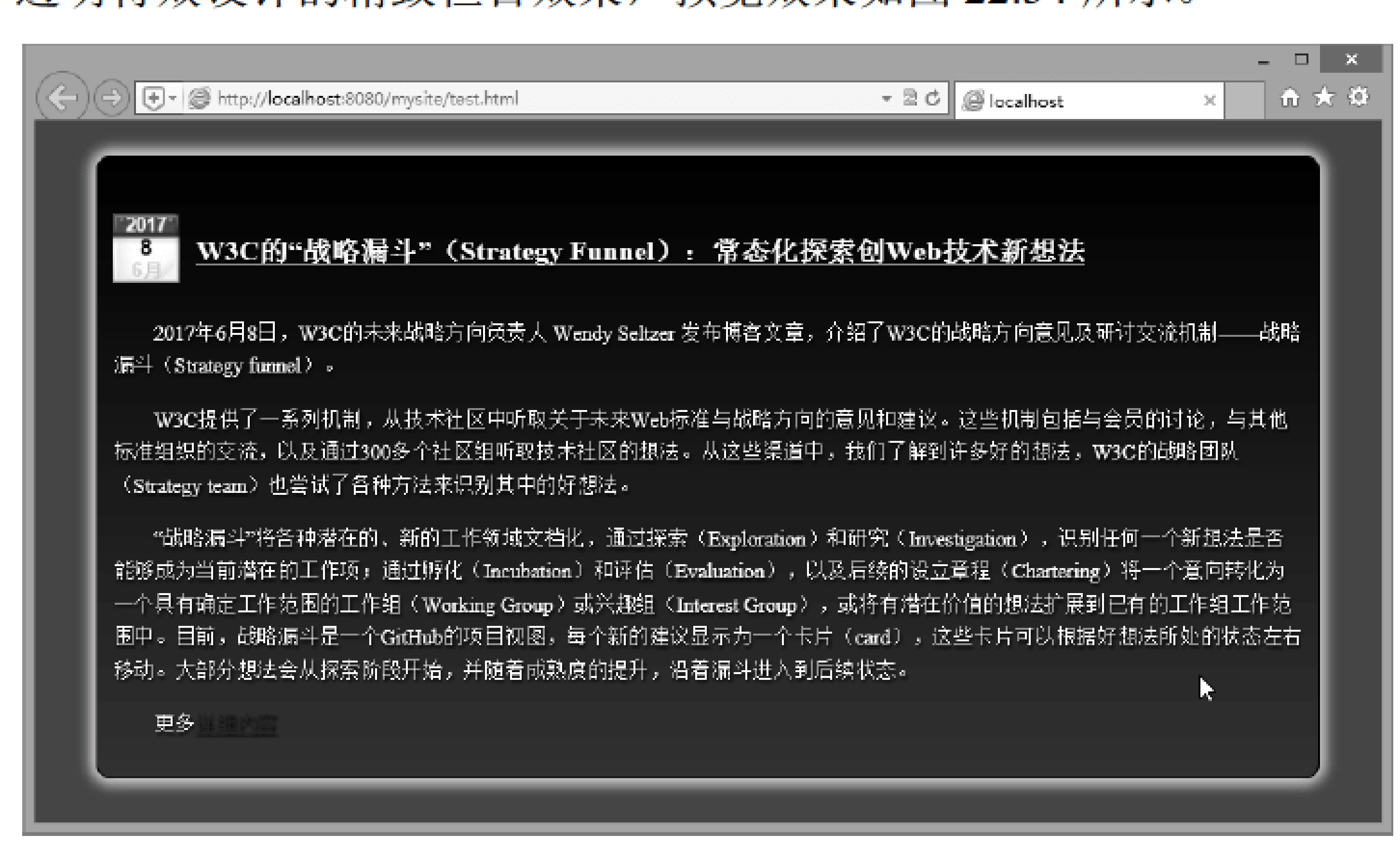


图 22.34 设计正文内容页

具体操作步骤请扫码学习。



视 频 讲 解

Note

22.5.2　设计应用界面

本例利用 CSS3 新增的边框和背景样式来模拟桌面界面效果。主要应用了 box-shadow、border-radius、text-shadow、border-color、border-image 等属性，同时还用到了渐变设计属性。整个案例的演示效果如图 22.35 所示。



图 22.35　设计 Windows 7 界面效果



线 上 阅 读

具体操作步骤请扫码学习。

22.6　在 线 练 习

本节提供了多个在线练习：（1）布局技巧；（2）排版方法；（3）CSS3 新特性。读者可通过这些专题练习 CSS3 的布局方法、特性和应用技巧。



在 线 练 习 1



在 线 练 习 2



在 线 练 习 3

第 23 章

CSS3 伸缩盒布局

2009 年，W3C 提出一种崭新的布局方案——伸缩盒布局，它可以简便、完整、响应式地实现各种页面布局，自由设置多个栏目在一个容器中的分布方式，以及如何处理容器内可用的空间。使用该模型可以轻松创建自适应窗口的流动布局，或者自适应字体大小的弹性布局。W3C 的伸缩盒布局分为旧版本、新版本，以及混合过渡版本三种不同的编码方式。其中混合过渡版本主要针对 IE 10 做了兼容。目前，该技术多应用在移动端网页布局，本章将主要讲解旧版本和新版本伸缩盒布局的基本用法。

权威参考：http://www.w3.org/TR/css-flexbox-1/



权 威 参 考

【学习重点】

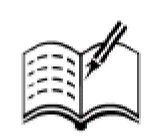
⏭ 设计多列布局。

⏭ 设计旧版伸缩盒布局。

⏭ 设计新版伸缩盒布局。



23.1　多 列 布 局



Note

CSS3 新增 columns 属性，用来设计多列布局，它允许网页内容跨栏显示，适合设计正文版式。权威参考：http://www.w3.org/TR/css3-multicol/。



权威参考　视频讲解

23.1.1　设置列宽

column-width 属性可以定义单列显示的宽度，基本语法如下所示：

```
column-width:<length> | auto
```

取值简单说明如下：

☑　<length>：用长度值来定义列宽。不允许负值。

☑　auto：根据<'column-count'>自定分配宽度，为默认值。

【示例】下面示例演示 column-width 属性在多列布局中的应用。设计 body 元素的列宽度为 300 像素，如果网页内容能够在单列内显示，则会以单列显示；如果窗口足够宽，且内容很多，则会在多列中进行显示，演示效果如图 23.1 所示，根据窗口宽度自动调整为两栏显示，列宽度显示为 300 像素。

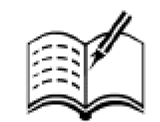


图 23.1　固定列表宽度显示

```
<style type="text/css">
/*定义网页列宽为 300 像素，则网页中每个栏目的最大宽度为 300 像素*/
body {column-width:300px;}
h1 {color: #333333; padding: 5px 8px;font-size: 20px;text-align: center; padding: 12px;}
h2 {font-size: 16px; text-align: center;}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
</style>
<h1>W3C 标准</h1>
<p>W3C 的各类技术标准在努力为各类应用的开发打造一个<strong>开放的 Web 平台（Open Web Platform）</strong>。尽管这个开放 Web 平台的边界在不断延伸，产业界认为 HTML5 将是这个平台的核心，平台的能力将依赖于 W3C 及其合作伙伴正在创建的一系列 Web 技术，包括 CSS, SVG, WOFF, 语义 Web，及 XML 和各类应用编程接口（APIs）。</p>
<p>截至 2014 年 3 月，W3C 共设立 5 个技术领域，开展 23 个标准计划。W3C 设有 46 个工作组（Working Group）、14 个兴趣小组（Interest Group）、3 个协调组（Coordination Group）、169 个社区组（Community Group），以及 3 个业务组（Business Group）。</p>
```



```
    <p>目前，W3C 正在探讨技术专家及个人参与 W3C 标准制定过程的 Webizen 计划，敬请期待。</p>
    <p>W3C 于 2014 年 11 月发布了题为&ldquo;W3C 工作重点（2014 年 11 月）"的报告，这是最新的一份对 W3C 近期开展的工作要点进行了综述的文章，阐述了近期的工作重点和优先级。</p>
```





视 频 讲 解

23.1.2　设置列数

column-count 属性可以定义显示的列数，基本语法如下所示：

```
column-count:<integer> | auto
```

取值简单说明如下：

☑　<integer>：用整数值来定义列数。不允许负值。

☑　auto：根据<' column-width '>自定分配宽度，为默认值。

【示例】在上面示例基础上，如果定义网页列数为 3，则不管浏览器窗口怎么调整，页面内容总是遵循 3 列布局，演示效果如图 23.2 所示。

```
/*定义网页列数为 3，这样整个页面总是显示为 3 列*/
body { column-count:3;}
```

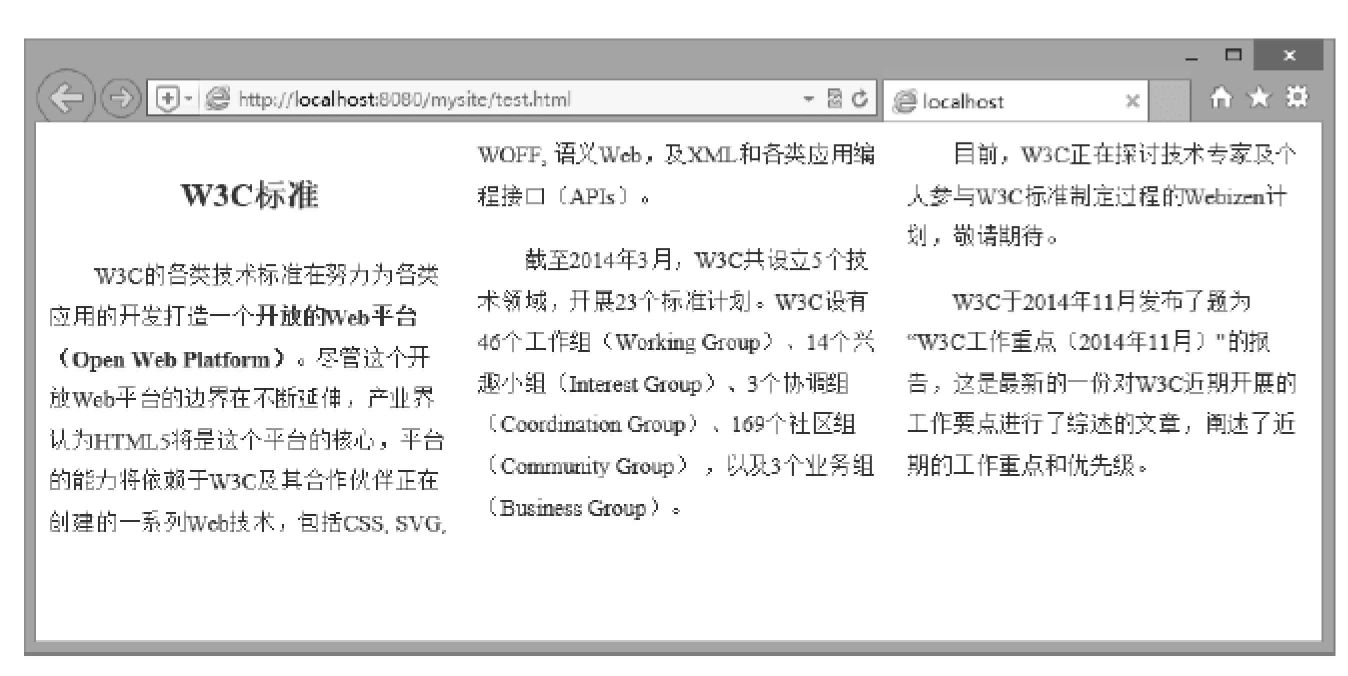


图 23.2　设计 3 列显示



视 频 讲 解

23.1.3　设置间距

column-gap 属性可以定义两栏之间的间距，基本语法如下所示：

```
column-gap:<length> | normal
```

取值简单说明如下：

☑　<length>：用长度值来定义列与列之间的间隙。不允许负值。

☑　normal：与<'font-size'>大小相同。如假设该对象的 font-size 为 16px，则 normal 值为 16px。

【示例】在上面示例基础上，通过 column-gap 和 line-height 属性配合使用，把文档版面设计得疏朗大方，以方便阅读。其中列间距为 3em，行高为 2.5em，页面内文字内容看起来更明晰、轻松，演示效果如图 23.3 所示。

```
body {
    /*定义页面内容显示为三列*/
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    column-gap: 3em;
```



```
        line-height: 2.5em; /* 定义页面文本行高 */
    }
```

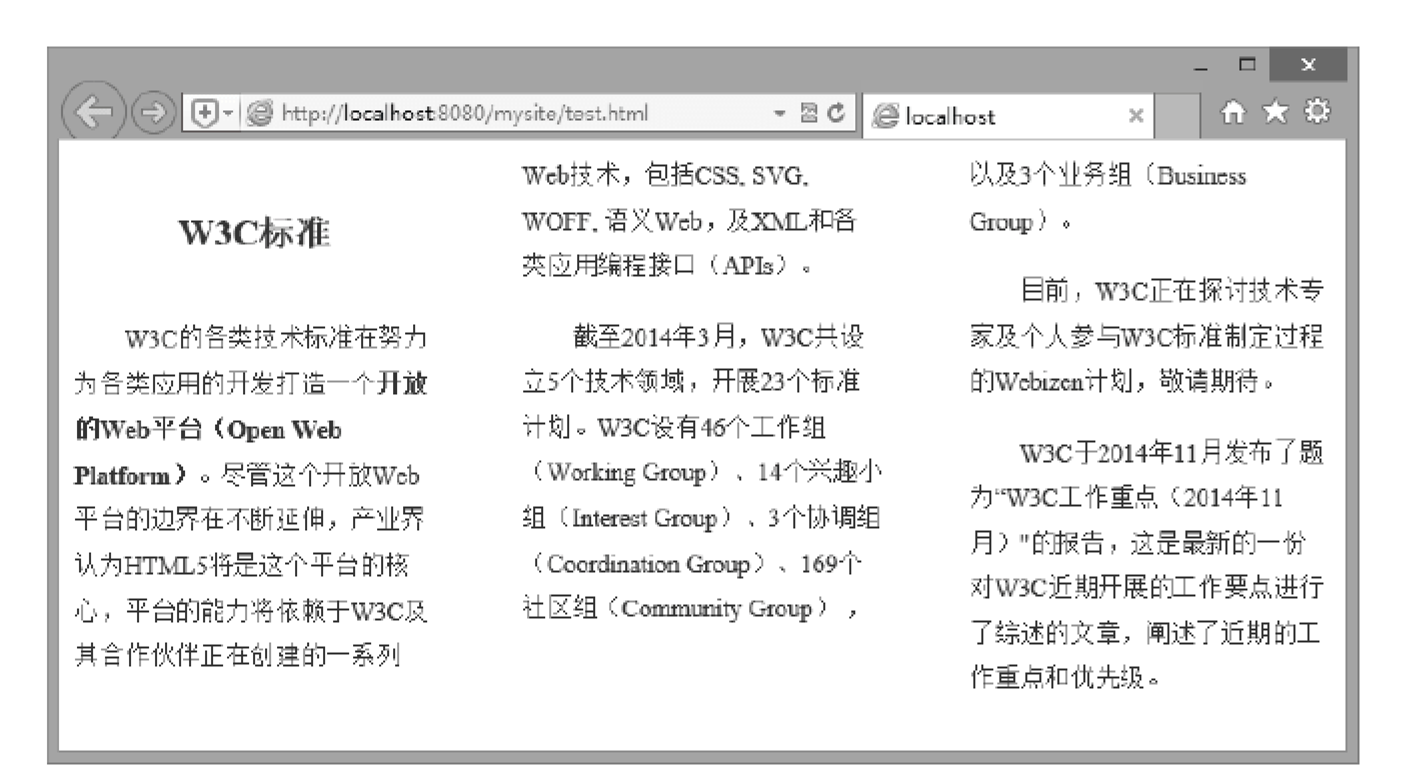
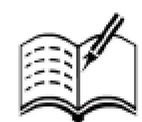


图 23.3　设计疏朗的跨栏布局



23.1.4　设置列边框



视 频 讲 解

column-rule 属性可以定义每列之间边框的宽度、样式和颜色。基本语法如下所示：

```
column-rule:<' column-rule-width '> || <' column-rule-style '> || <' column-rule-color '>
```

取值简单说明如下：

☑　<' column-rule-width '>：设置对象的列与列之间的边框厚度。

☑　<' column-rule-style '>：设置对象的列与列之间的边框样式。

☑　<' column-rule-color '>：设置对象的列与列之间的边框颜色。

column-rule-style 属性语法如下所示（取值与边框样式 border-style 相同）：

```
column-rule-style:none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset
```

column-rule-width 与 border-width，column-rule-color 与 border-color 设置相同。

【示例】在上面示例基础上，为每列之间的边框定义一个虚线分割线，线宽为 2 像素，灰色显示，演示效果如图 23.4 所示。

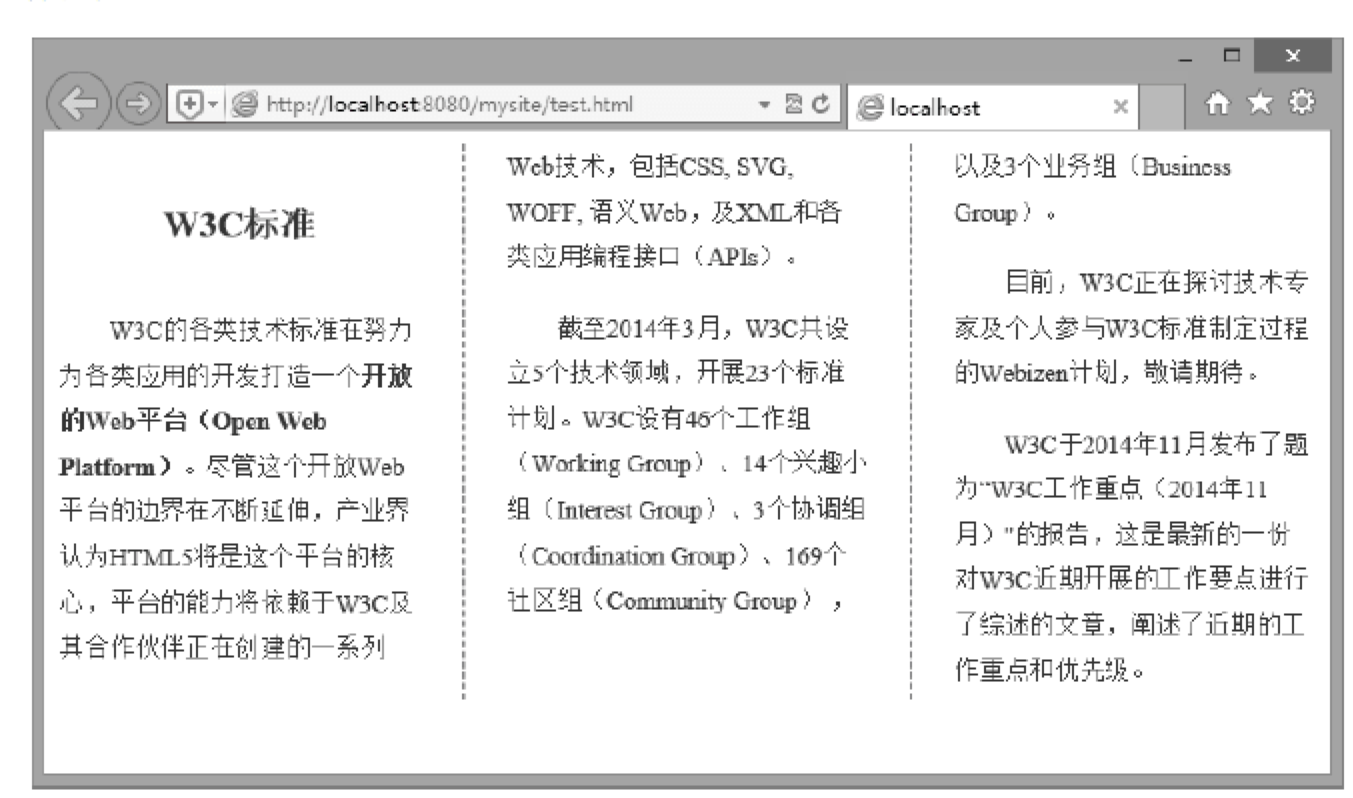
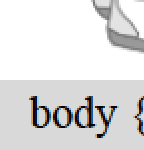


图 23.4　设计列边框效果

```
body {
    /*定义页面内容显示为三列*/
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    column-gap: 3em;
    line-height: 2.5em;
    /*定义列边框为 2 像素宽的灰色虚线*/
    column-rule: dashed 2px gray;
}
```



Note

视频讲解

23.1.5 设置跨列显示

column-span 属性可以定义跨列显示，基本语法如下所示：

```
column-span:none | all
```

取值简单说明如下：

☑ none：不跨列。

☑ all：横跨所有列。

【示例】在上面示例基础上，使用 column-span 属性定义一级标题跨列显示，演示效果如图 23.5 所示。

```
body {
    /*定义页面内容显示为三列*/
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    column-gap: 3em;
    line-height: 2.5em;
    /*定义列边框为 2 像素宽的灰色虚线*/
    column-rule: dashed 2px gray;}
/*设置一级标题跨越所有列显示*/
h1 {
    color: #333333; font-size: 20px; text-align: center;
    padding: 12px;
    /*跨越所有列显示*/
    column-span: all;
}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
```

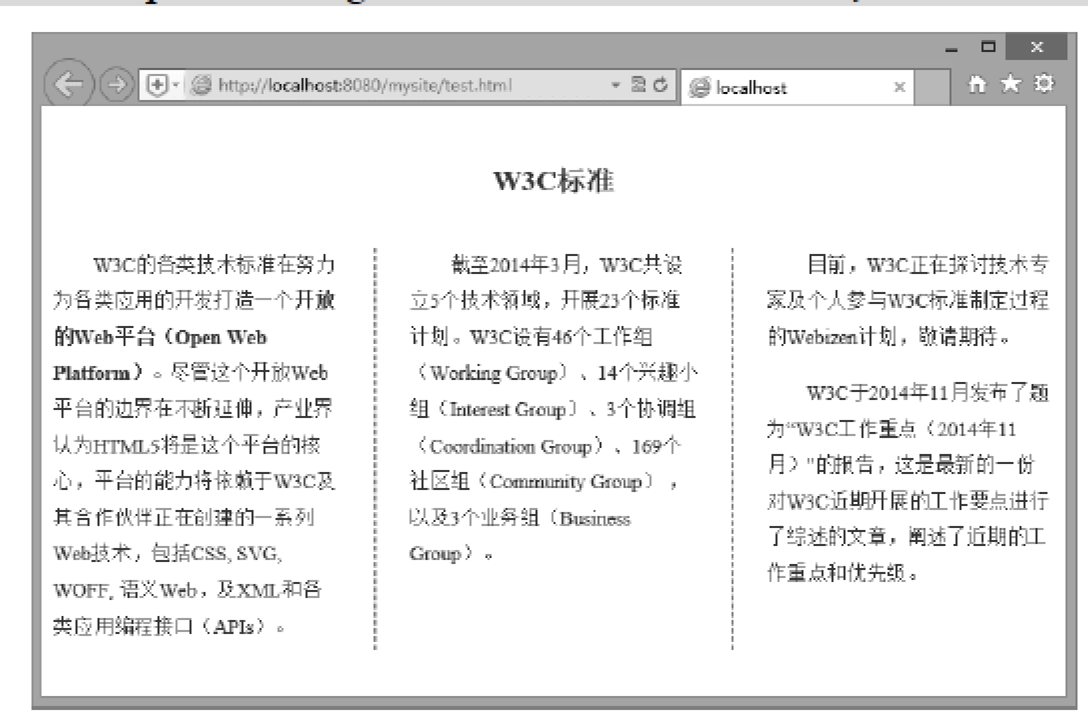
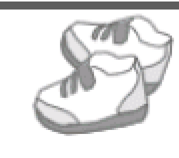


图 23.5 设计标题跨列显示效果





23.1.6　设置列高度

column-fill 属性可以定义栏目的高度是否统一，基本语法如下所示：

```
column-fill:auto | balance
```

取值简单说明如下：

☑　auto：列高度自适应内容。

☑　balance：所有列的高度与其中最高的一列统一。

【示例】在上面示例基础上，使用 column-fill 属性定义每列高度一致。

```
body {
    /*定义页面内容显示为三列*/
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    column-gap: 3em;
    line-height: 2.5em;
    /*定义列边框为 2 像素宽的灰色虚线*/
    column-rule: dashed 2px gray;
    /*设置各列高度一致*/
    column-fill: balance;
}
```

23.2　旧版伸缩盒

Flexbox（伸缩盒）是 CSS3 新增的布局模型，实际上它一直都存在。最开始它作为 Mozilla XUL 的一个功能被用来制作程序界面，如 Firefox 的工具栏，就多次使用这个属性。本节将重点介绍旧版本伸缩盒模型的基本用法。

23.2.1　启动伸缩盒

在旧版本中启动伸缩盒模型，只需设置容器的 display 的属性值为 box（或 inline-box），用法如下所示：

```
display: box;
display: inline-box;
```

伸缩盒模型包括两部分：

☑　父容器。

☑　子容器。

父容器通过“display:box;”或者“display: inline-box;”启动伸缩盒布局功能。

子容器通过 box-flex 属性定义布局宽度，定义如何对父容器的宽度进行分配。

父容器又通过如下属性定义包含容器的显示属性，简单说明如下：

☑　box-orient：定义父容器里子容器的排列方式，是水平还是垂直。

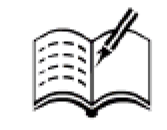
☑　box-direction：定义父容器里子容器的排列顺序。

☑　box-align：定义子容器的垂直对齐方式。

☑　box-pack：定义子容器的水平对齐方式。



> 注意：使用旧版本伸缩盒模型，需要用到各浏览器的私有属性，Webkit 引擎支持-webkit-前缀的私有属性，Mozilla Gecko 引擎支持-moz-前缀的私有属性，Presto 引擎（包括 Opera 浏览器等）支持标准属性，IE 暂不支持旧版本伸缩盒模型。



Note



视频讲解

23.2.2 设置宽度

在默认情况下，盒子没有弹性，它将尽可能宽地使其内容可见，且没有溢出，其大小由 width、height、min-height、min-width、max-width 或者 max-height 属性值来决定。

使用 box-flex 属性可以把默认布局变为盒布局。如果 box-flex 的属性值为 1，则元素变得富有弹性，其大小将按下面的方式计算：

- ☑ 声明的大小（width、height、min-width、min-height、max-width、max-height）。
- ☑ 父容器的大小和所有余下的可利用的内部空间。

如果盒子没有声明大小，那么其大小将完全取决于父容器的大小，即盒子的大小等于父容器的大小乘以其 box-flex 在所有盒子 box-flex 总和中的百分比，用公式表示：

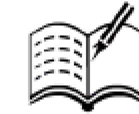
```
盒子的大小 = 父容器的大小 × 盒子的 box-flex / 所有盒子的 box-flex 值的和
```

余下的盒子将按照上面的原则分享剩下的可用空间。

【示例】下面示例定义左侧边栏的宽度为 240px，右侧边栏的宽度为 200px，中间内容版块的宽度将由 box-flex 属性确定。详细代码如下所示：

```
<style type="text/css">
#container {
    /*定义弹性盒布局样式*/
    display: -moz-box;
    display: -webkit-box;
    display: box;
}
#left-sidebar {
    width: 240px;
    padding: 20px;
    background-color: orange;
}
#contents {
    /*定义中间列宽度为自适应显示*/
    -moz-box-flex: 1;
    -webkit-box-flex: 1;
    flex: 1;
    padding: 20px;
    background-color: yellow;
}
#right-sidebar {
    width: 200px;
    padding: 20px;
    background-color: limegreen;
}
#left-sidebar, #contents, #right-sidebar {
    /*定义盒样式*/
    -moz-box-sizing: border-box;
```





Note

```
            -webkit-box-sizing: border-box;
            box-sizing: border-box;
        }
    </style>
    <div id="container">
        <div id="left-sidebar">
            <h2>宋词精选</h2>
            <ul>
                <li><a href="">卜算子 • 咏梅</a></li>
                <li><a href=""> 声声慢 • 寻寻觅觅</a></li>
                <li><a href=""> 雨霖铃 • 寒蝉凄切</a></li>
                <li><a href="">卜算子 • 咏梅</a></li>
                <li><a href="">更多</a></li>
            </ul>
        </div>
        <div id="contents">
            <h1>水调歌头 • 明月几时有</h1>
            <h2>苏轼</h2>
            <p>丙辰中秋，欢饮达旦，大醉，作此篇，兼怀子由。</p>
            <p>明月几时有？把酒问青天。不知天上宫阙，今夕是何年。我欲乘风归去，又恐琼楼玉宇，高处
不胜寒。起舞弄清影，何似在人间？</p>
            <p>转朱阁，低绮户，照无眠。不应有恨，何事长向别时圆？人有悲欢离合，月有阴晴圆缺，此事
古难全。但愿人长久，千里共婵娟。</p>
        </div>
        <div id="right-sidebar">
            <h2>词人列表</h2>
            <ul>
                <li><a href="">陆游</a></li>
                <li><a href="">李清照</a></li>
                <li><a href="">苏轼</a></li>
                <li><a href="">柳永</a></li>
            </ul>
        </div>
    </div>
```

演示效果如图 23.6 所示，当调整窗口宽度时，中间列的宽度会自适应显示，使整个页面总是满窗口显示。



图 23.6　定义自适应宽度





视频讲解

Note

23.2.3　设置顺序

使用 box-ordinal-group 属性可以改变子元素的显示顺序。语法格式如下：

```
box-ordinal-group:<integer>
```

<integer>用整数值来定义伸缩盒对象的子元素显示顺序，默认值为 1。浏览器在显示时，将根据该值从小到大来显示这些元素。

【示例】以 23.2.2 节示例为基础，在左栏、中栏、右栏中分别加入一个 box-ordinal-group 属性，并指定显示的序号，这里将中栏设置为 1，右栏设置为 2，左栏设置为 3，则可以发现三栏显示顺序发生了变化，演示效果如图 23.7 所示。

```
#left-sidebar {
    -moz-box-ordinal-group: 3;
    -webkit-box-ordinal-group: 3;
    box-ordinal-group: 3;
}
#contents {
    -moz-box-ordinal-group: 1;
    -webkit-box-ordinal-group: 1;
    box-ordinal-group: 1;
}
#right-sidebar {
    -moz-box-ordinal-group: 2;
    -webkit-box-ordinal-group: 2;
    box-ordinal-group: 2;
}
```



图 23.7　定义列显示顺序



视频讲解

23.2.4　设置方向

使用 box-orient 可以定义元素的排列方向，语法格式如下所示：

```
box-orient:horizontal | vertical | inline-axis | block-axis
```

取值简单说明如下：

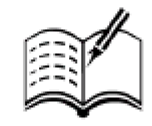
☑　horizontal：设置伸缩盒对象的子元素从左到右水平排列。



☑ vertical：设置伸缩盒对象的子元素从上到下纵向排列。

☑ inline-axis：设置伸缩盒对象的子元素沿行轴排列。

☑ block-axis：设置伸缩盒对象的子元素沿块轴排列。



【示例】针对上面示例，在<div id="container">标签样式中加入 box-orient 属性，并设定属性值为 vertical，即定义内容以垂直方向排列，则代表左侧边栏、中间内容、右侧边栏的三个 div 元素的排列方向将从水平方向改变为垂直方向，演示效果如图 23.8 所示。

```
#container {
    /*定义弹性盒布局样式*/
    display: -moz-box;
    display: -webkit-box;
    display: box;
    /*定义从上到下排列显示*/
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    box-orient: vertical;
}
```



图 23.8　定义列显示方向

使用 box-direction 属性可以让各个子元素反向排序，语法格式如下：

```
box-direction:normal | reverse
```

取值简单说明如下：

☑ normal：设置伸缩盒对象的子元素按正常顺序排列。

☑ reverse：反转伸缩盒对象的子元素的排列顺序。

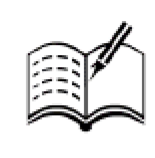
23.2.5　设置对齐方式



使用 box-pack 可以设置子元素水平方向对齐方式，语法格式如下：

```
box-pack:start | center | end | justify
```

取值简单说明如下：




☑ start：设置伸缩盒对象的子元素从开始位置对齐，为默认值。
☑ center：设置伸缩盒对象的子元素居中对齐。
☑ end：设置伸缩盒对象的子元素从结束位置对齐。
☑ justify：设置或伸缩盒对象的子元素两端对齐。

使用 box-align 可以设置子元素垂直方向对齐方式，语法格式如下：

```
box-align:start | end | center | baseline | stretch
```

取值简单说明如下：

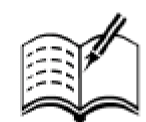
☑ start：设置伸缩盒对象的子元素从开始位置对齐。
☑ end：设置伸缩盒对象的子元素从结束位置对齐。
☑ center：设置伸缩盒对象的子元素居中对齐。
☑ baseline：设置伸缩盒对象的子元素基线对齐。
☑ stretch：设置伸缩盒对象的子元素自适应父元素尺寸。

【示例】在下面示例中有一个<div class="login">容器，其中包含一个登录表单对象，为了方便练习，本例使用一个<img>标签模拟，然后使用 box-pack 和 box-align 属性让表单对象在<div class="login">容器的正中央显示。同时，设计<div class="login">容器高度和宽度都为 100%，这样就可以让表单对象在窗口中央位置显示，具体实现代码如下：

```
<style type="text/css">
/*清除页边距*/
body { margin: 0; padding: 0;}
div { position: absolute; }
.bg {/*设计遮罩层*/
    width: 100%; height: 100%;
    background: #000; opacity: 0.7;
}
.login {
    /*满屏显示*/
    width:100%; height:100%;
    /*定义弹性盒布局样式*/
    display: -moz-box;
    display: -webkit-box;
    display: box;
    /*垂直居中显示*/
    -moz-box-align: center;
    -webkit-box-align: center;
    box-align: center;
    /*水平居中显示*/
    -moz-box-pack: center;
    -webkit-box-pack: center;
    box-pack: center;
}
</style>
<div class="web"><img src="images/bg.png" /></div>
<div class="bg"></div>
<div class="login"><img src="images/login.png"   /></div>
```

设计效果如图 23.9 所示。





Note





图 23.9　设计登录表单中央显示

23.3　新版伸缩盒

伸缩盒模型是一个新的盒子模型，它主要优化了 UI 布局，可以简单地使一个元素居中（包括水平和垂直居中），可以扩大或收缩元素来填充容器的可利用空间，可以改变布局顺序等。本节将重点介绍新版本伸缩盒模型的基本用法。

23.3.1　认识 Flexbox 系统

Flexbox 由伸缩容器和伸缩项目组成。

在伸缩容器中，每一个子元素都是一个伸缩项目，伸缩项目可以是任意数量的，伸缩容器外和伸缩项目内的一切元素都不受影响。

伸缩项目沿着伸缩容器内的一个伸缩行定位，通常每个伸缩容器只有一个伸缩行。在默认情况下，伸缩行和文本方向一致：从左至右，从上到下。

常规布局是基于块和文本流方向，而 Flex 布局是基于 flex-flow 流。如图 23.10 所示是 W3C 规范对 Flex 布局的解释。

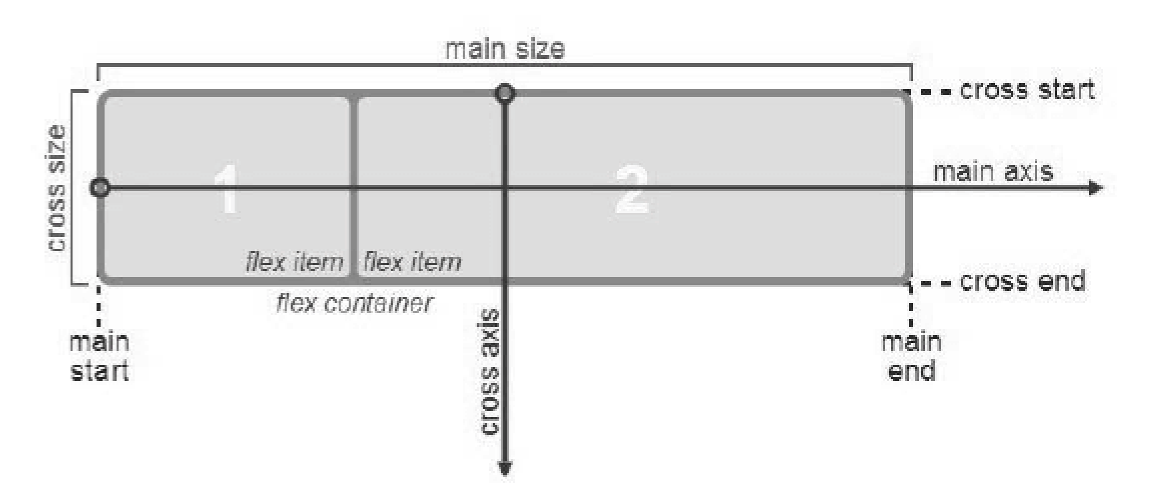


图 23.10　Flex 布局模式

伸缩项目是沿着主轴（main axis），从主轴起点（main-start）到主轴终点（main-end），或者沿着侧轴（cross axis），从侧轴起点（cross-start）到侧轴终点（cross-end）排列。



Note

- ☑ 主轴（main axis）：伸缩容器的主轴，伸缩项目主要沿着这条轴进行排列布局。注意，它不一定是水平的，这主要取决于 justify-content 属性设置。
- ☑ 主轴起点（main-start）和主轴终点（main-end）：伸缩项目放置在伸缩容器内从主轴起点（main-start）向主轴终点（main-end）方向。
- ☑ 主轴尺寸（main size）：伸缩项目在主轴方向的宽度或高度就是主轴的尺寸。伸缩项目主要的大小属性要么是宽度，要么是高度属性，由哪一个对着主轴方向决定。
- ☑ 侧轴（cross axis）：垂直于主轴称为侧轴。它的方向主要取决于主轴方向。
- ☑ 侧轴起点（cross-start）和侧轴终点（cross-end）：伸缩行的配置从容器的侧轴起点边开始，往侧轴终点边结束。
- ☑ 侧轴尺寸（cross size）：伸缩项目在侧轴方向的宽度或高度就是项目的侧轴长度，伸缩项目的侧轴长度属性是 width 或 height 属性，由哪一个对着侧轴方向决定。

一个伸缩项目就是一个伸缩容器的子元素，伸缩容器中的文本也被视为一个伸缩项目。伸缩项目中的内容与普通文本流一样。例如，当一个伸缩项目被设置为浮动，用户依然可以在这个伸缩项目中放置一个浮动元素。



视频讲解

23.3.2 启动伸缩盒

通过设置元素的 display 属性为 flex 或 inline-flex 可以定义一个伸缩容器。设置为 flex 的容器被渲染为一个块级元素，而设置为 inline-flex 的容器则渲染为一个行内元素。具体语法如下：

```
display: flex | inline-flex;
```

上面语法定义伸缩容器，属性值决定容器是行内显示，还是块显示，它的所有子元素将变成 flex 文档流，被称为伸缩项目。

此时，CSS 的 columns 属性在伸缩容器上没有效果，同时 float、clear 和 vertical-align 属性在伸缩项目上也没有效果。

【示例】下面示例设计一个伸缩容器，其中包含四个伸缩项目，演示效果如图 23.11 所示。

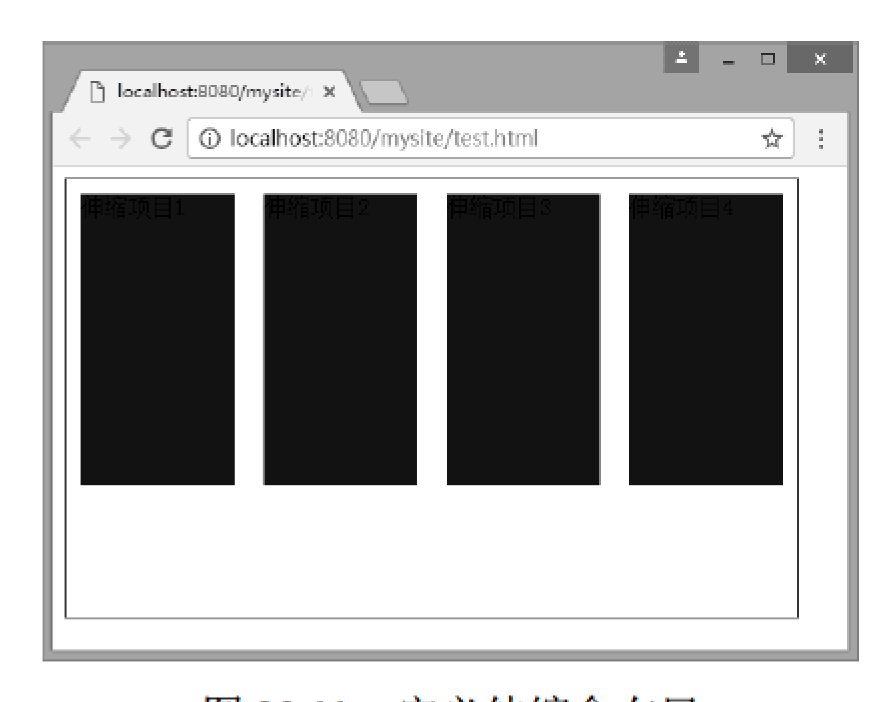
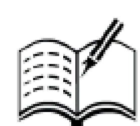


图 23.11　定义伸缩盒布局

```
<style type="text/css">
.flex-container {
    display: -webkit-flex;
    display: flex;
    width: 500px; height: 300px;
    border: solid 1px red;
```



```
    }
    .flex-item {
        background-color: blue;
        width: 200px; height: 200px;
        margin: 10px;
    }
    </style>
    <div class="flex-container">
        <div class="flex-item">伸缩项目 1</div>
        <div class="flex-item">伸缩项目 2</div>
        <div class="flex-item">伸缩项目 3</div>
        <div class="flex-item">伸缩项目 4</div>
    </div>
```



Note

23.3.3　设置主轴方向



视频讲解

使用 flex-direction 属性可以定义主轴方向，它适用于伸缩容器。具体语法如下：

```
flex-direction:row | row-reverse | column | column-reverse
```

取值说明如下：

- ☑ row：主轴与行内轴方向作为默认的书写模式。即横向从左到右排列（左对齐）。
- ☑ row-reverse：对齐方式与 row 相反。
- ☑ column：主轴与块轴方向作为默认的书写模式。即纵向从上往下排列（顶对齐）。
- ☑ column-reverse：对齐方式与 column 相反。

【示例】在 23.3.2 节示例基础上，本例设计一个伸缩容器，其中包含四个伸缩项目，然后定义伸缩项目从上往下排列，演示效果如图 23.12 所示。

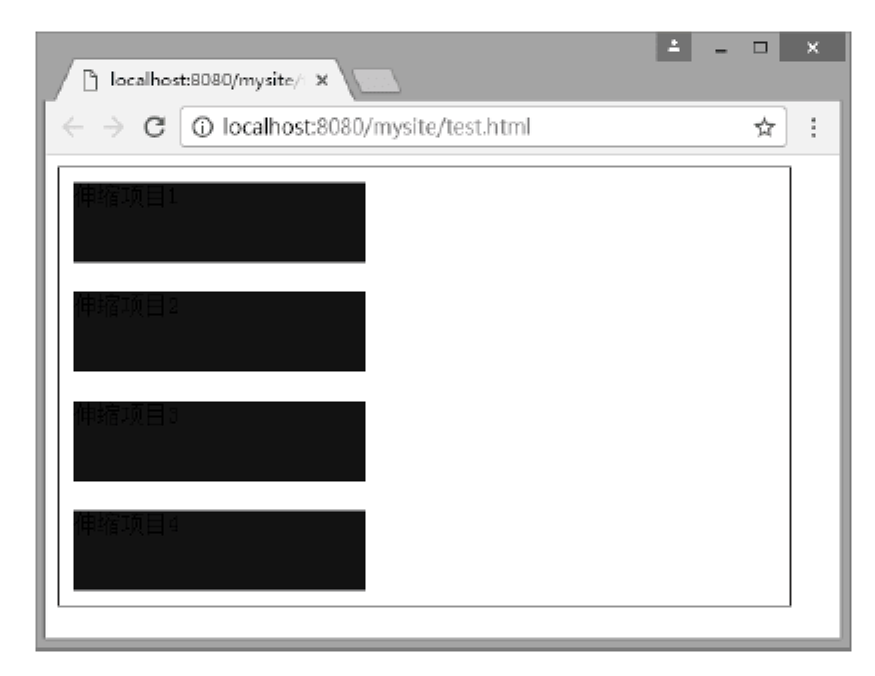
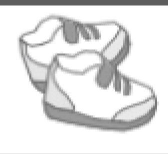
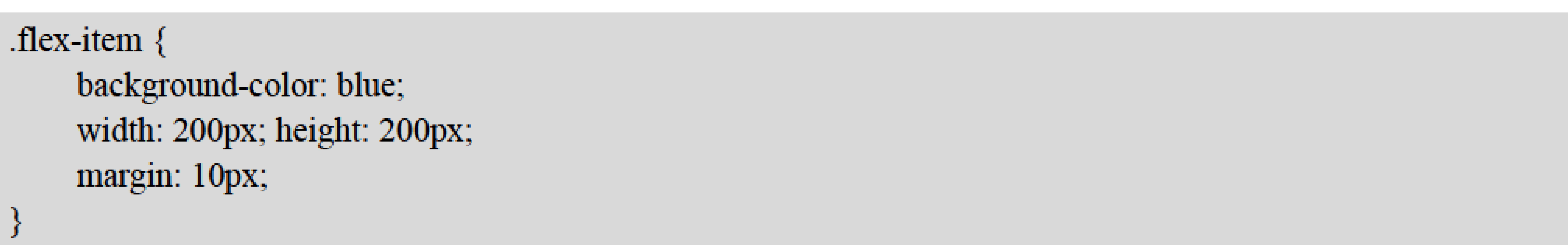


图 23.12　定义伸缩项目从上往下布局

```
    <style type="text/css">
    .flex-container {
        display: -webkit-flex;
        display: flex;
        -webkit-flex-direction: column;
        flex-direction: column;
        width: 500px;height: 300px;border: solid 1px red;
    }
```





```
.flex-item {
    background-color: blue;
    width: 200px; height: 200px;
    margin: 10px;
}
</style>
```



Note



视频讲解

23.3.4 设置行数

flex-wrap 定义伸缩容器是单行还是多行显示伸缩项目，侧轴的方向决定了新行堆放的方向。具体语法格式如下：

```
flex-wrap:nowrap | wrap | wrap-reverse
```

取值说明如下：

- ☑ nowrap：flex 容器为单行。该情况下 flex 子项可能会溢出容器。
- ☑ wrap： flex 容器为多行。该情况下 flex 子项溢出的部分会被放置到新行，子项内部会发生断行。
- ☑ wrap-reverse：反转 wrap 排列。

【示例】在上面示例基础上，下面示例设计一个伸缩容器，其中包含四个伸缩项目，然后定义伸缩项目多行排列，演示效果如图 23.13 所示。

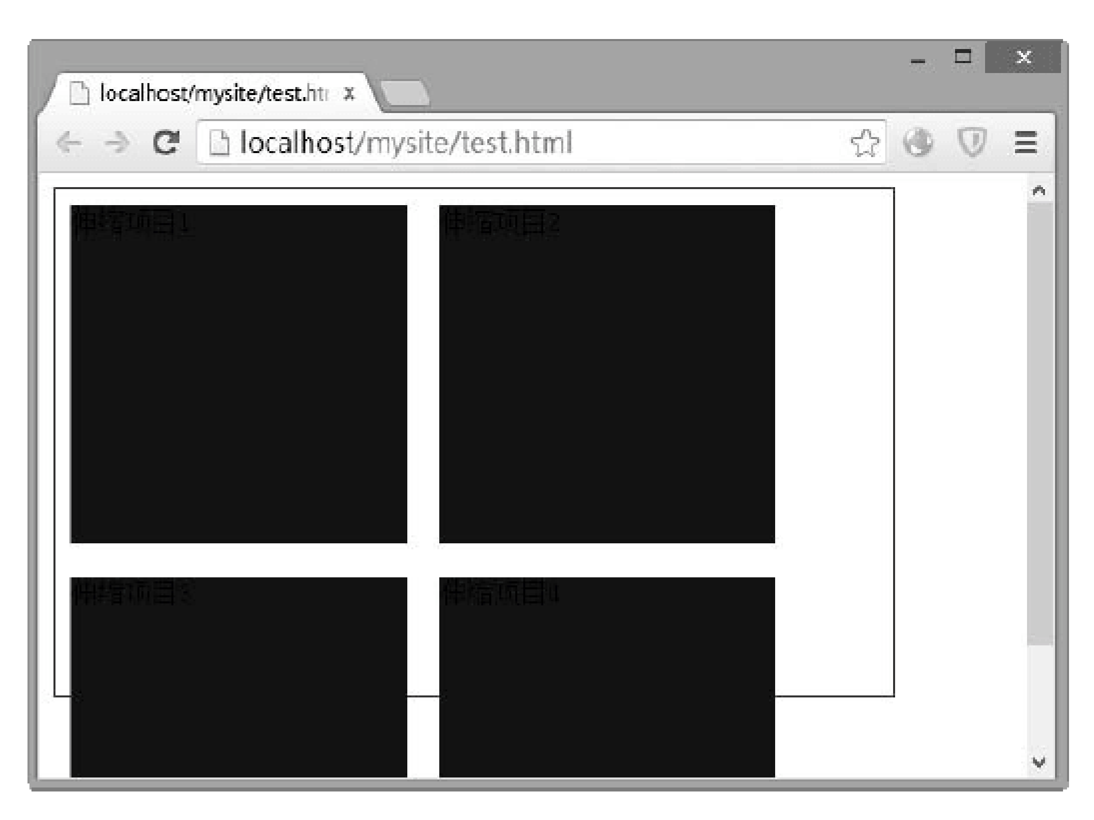


图 23.13　定义伸缩项目多行布局

```
<style type="text/css">
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-wrap: wrap;
    flex-wrap: wrap;
    width: 500px; height: 300px;border: solid 1px red;
}
.flex-item {
    background-color: blue;
    width: 200px; height: 200px;
    margin: 10px;
```

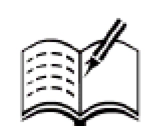


```
}
</style>
```

【补充】

flex-flow 属性是 flex-direction 和 flex-wrap 属性的复合属性，适用于伸缩容器。该属性可以同时定义伸缩容器的主轴和侧轴。其默认值为 row nowrap。具体语法如下：

```
flex-flow:<' flex-direction '> || <' flex-wrap '>
```



取值说明如下：

- ☑ <' flex-direction'>：定义弹性盒子元素的排列方向。
- ☑ <' flex-wrap'>：控制 flex 容器是单行或者多行。

23.3.5　设置对齐方式



1．主轴对齐

justify-content 定义伸缩项目沿着主轴线的对齐方式，该属性适用于伸缩容器。具体语法如下：

```
justify-content:flex-start | flex-end | center | space-between | space-around
```

取值说明如下：

- ☑ flex-start：为默认值，伸缩项目向一行的起始位置靠齐。
- ☑ flex-end：伸缩项目向一行的结束位置靠齐。
- ☑ center：伸缩项目向一行的中间位置靠齐。
- ☑ space-between：伸缩项目会平均地分布在行里。第一个伸缩项目在一行中的最开始位置，最后一个伸缩项目在一行中最终点位置。
- ☑ space-around：伸缩项目会平均地分布在行里，两端保留一半的空间。

上述取值比较效果如图 23.14 所示。

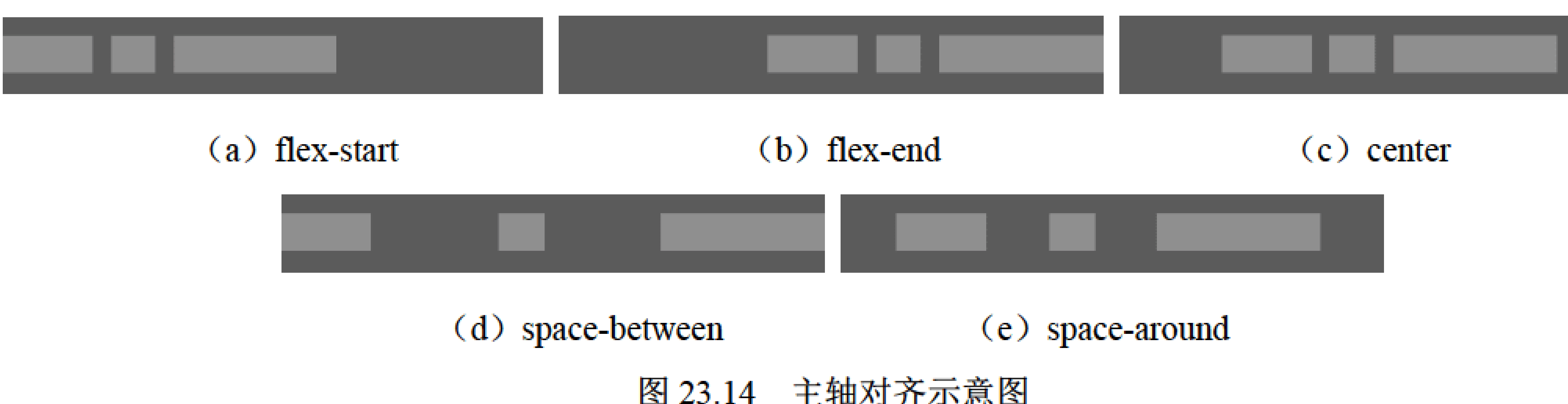


图 23.14　主轴对齐示意图

2．侧轴对齐

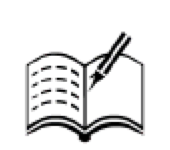
align-items 定义伸缩项目在侧轴上的对齐方式，该属性适用于伸缩容器。具体语法如下：

```
align-items:flex-start | flex-end | center | baseline | stretch
```

取值说明如下：

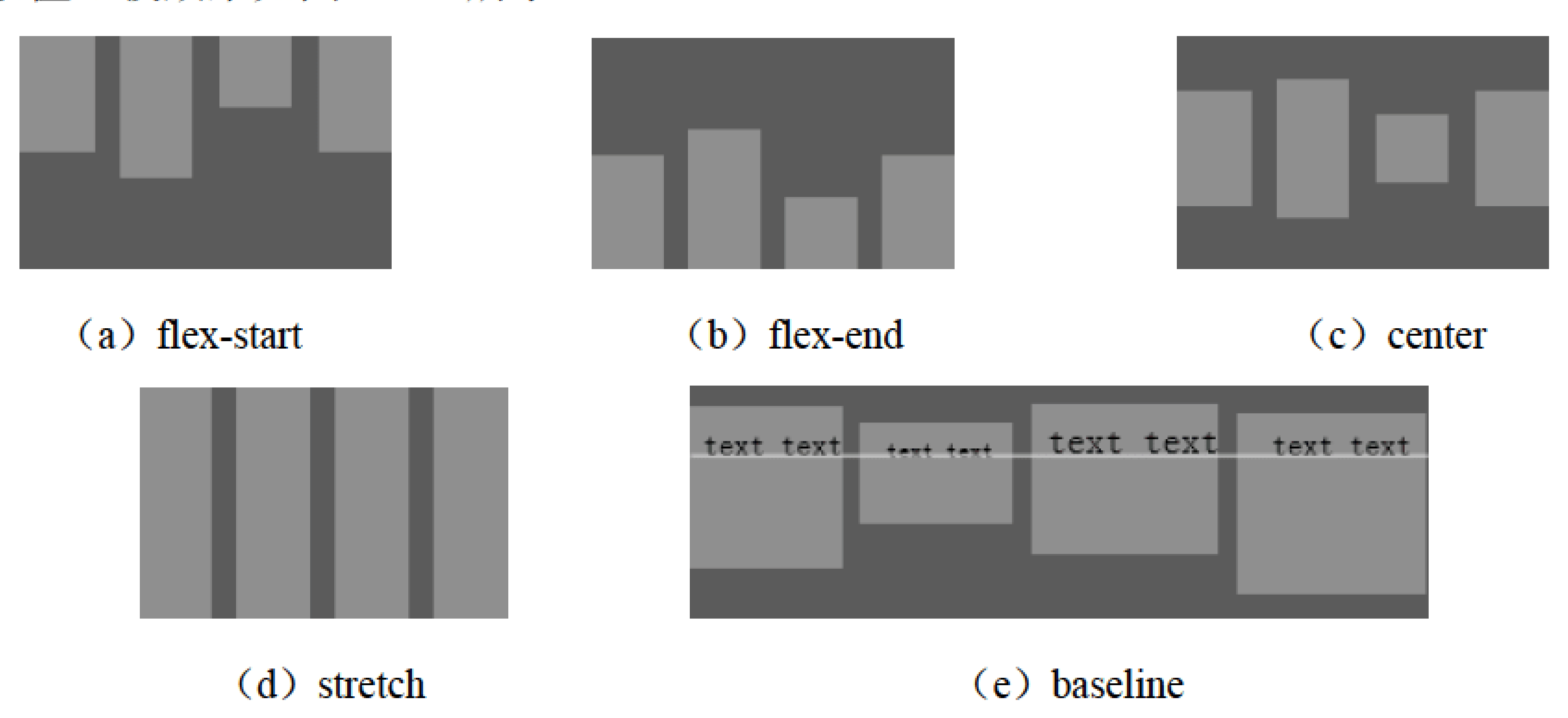
- ☑ flex-start：伸缩项目在侧轴起点边的外边距紧靠住该行在侧轴起始的边。
- ☑ flex-end：伸缩项目在侧轴终点边的外边距紧靠住该行在侧轴终点的边 。
- ☑ center：伸缩项目的外边距盒在该行的侧轴上居中放置。
- ☑ baseline：伸缩项目根据它们的基线对齐。
- ☑ stretch：默认值，伸缩项目拉伸填充整个伸缩容器。此值会使项目的外边距盒的尺寸在遵照 min/max-width/height 属性的限制下尽可能接近所在行的尺寸。





Note

上述取值比较效果如图 23.15 所示。



（a）flex-start　（b）flex-end　（c）center

（d）stretch　（e）baseline

图 23.15　侧轴对齐示意图

3．伸缩行对齐

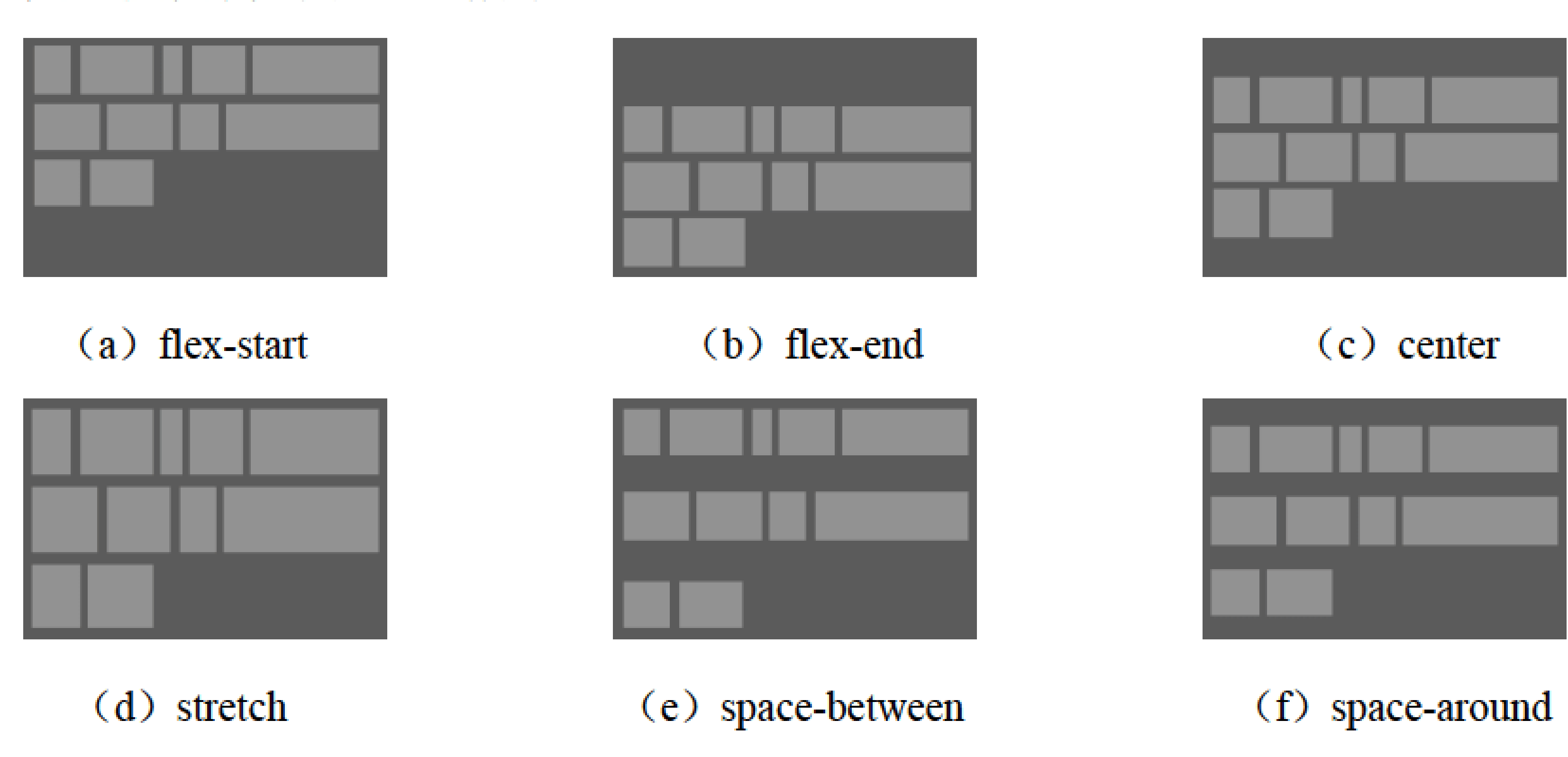
align-content 定义伸缩行在伸缩容器里的对齐方式，该属性适用于伸缩容器。类似于伸缩项目在主轴上使用 justify-content 属性一样，但本属性在只有一行的伸缩容器上没有效果。具体语法如下：

```
align-content:flex-start | flex-end | center | space-between | space-around | stretch
```

取值说明如下：

☑ flex-start：各行向伸缩容器的起点位置堆叠。
☑ flex-end：各行向伸缩容器的结束位置堆叠。
☑ center：各行向伸缩容器的中间位置堆叠。
☑ space-between：各行在伸缩容器中平均分布。
☑ space-around：各行在伸缩容器中平均分布，在两边各有一半的空间。
☑ stretch：默认值，各行将会伸展以占用剩余的空间。

上述取值比较效果如图 23.16 所示。



（a）flex-start　（b）flex-end　（c）center

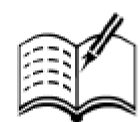
（d）stretch　（e）space-between　（f）space-around

图 23.16　伸缩行对齐示意图

【示例】下面示例以上面示例为基础，定义伸缩行在伸缩容器中居中显示，演示效果如图 23.17 所示。

```
<style type="text/css">
```





Note

```
.flex-container {
    display: -webkit-flex;
    display: flex;
    -webkit-flex-wrap: wrap;
    flex-wrap: wrap;
    -webkit-align-content: center;
    align-content: center;
    width: 500px; height: 300px;border: solid 1px red;
}
.flex-item {
    background-color: blue;
    width: 200px; height: 200px;
    margin: 10px;
}
</style>
```

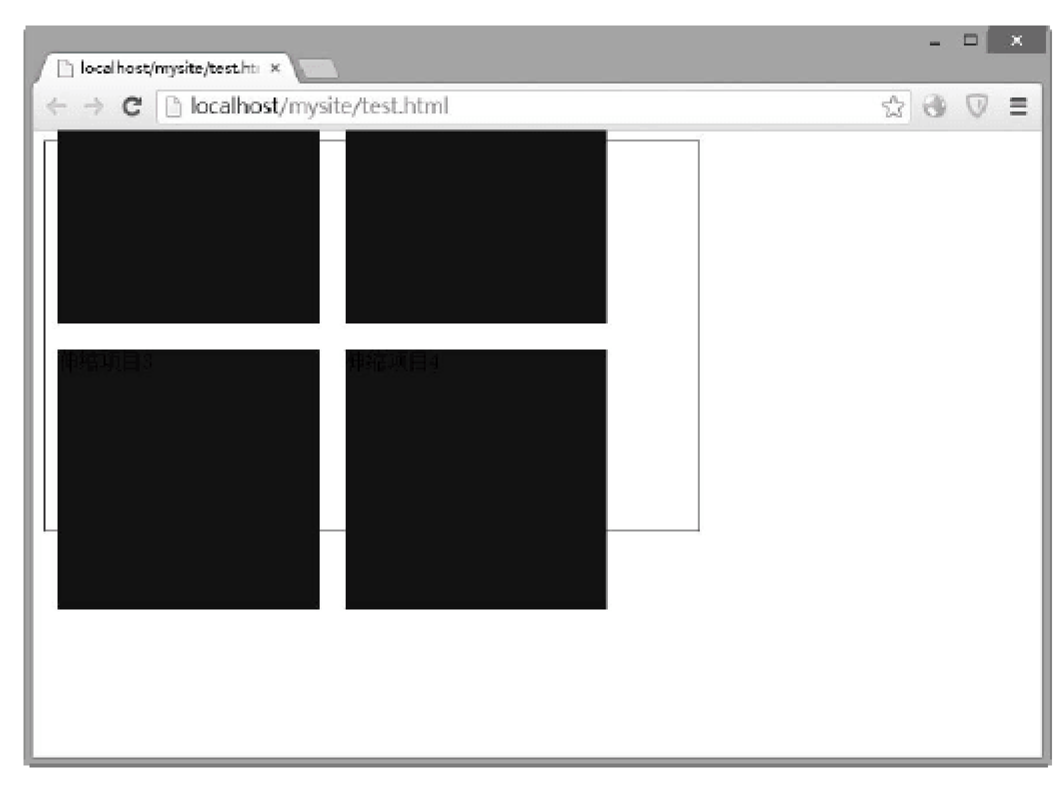


图 23.17　定义伸缩行居中对齐

23.3.6　设置伸缩项目



视 频 讲 解

伸缩项目都有一个主轴长度（Main Size）和一个侧轴长度（Cross Size）。主轴长度是伸缩项目在主轴上的尺寸，侧轴长度是伸缩项目在侧轴上的尺寸。一个伸缩项目的宽或高取决于伸缩容器的轴，可能就是它的主轴长度或侧轴长度。下面属性适用于伸缩项目，可以调整伸缩项目的行为。

1．显示位置

order 属性可以控制伸缩项目在伸缩容器中的显示顺序，具体语法如下：

```
order: <integer>
```

<integer>用整数值来定义排列顺序，数值小的排在前面。可以为负值。

2．扩展空间

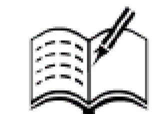
flex-grow 可以定义伸缩项目的扩展能力，决定伸缩容器剩余空间按比例应扩展多少空间。具体语法如下：

```
flex-grow: <number>
```

<number>用数值来定义扩展比率。不允许负值，默认值为 0。



如果所有伸缩项目的 flex-grow 设置为 1，那么每个伸缩项目将设置为一个大小相等的剩余空间。如果给其中一个伸缩项目设置 flex-grow 为 2，那么这个伸缩项目所占的剩余空间是其他伸缩项目所占剩余空间的两倍。



Note

3．收缩空间

flex-shrink 可以定义伸缩项目收缩的能力，与 flex-grow 功能相反，具体语法如下：

```
flex-shrink: <number>
```

<number>用数值来定义收缩比率。不允许负值，默认值为 1。

4．伸缩比率

flex-basis 可以设置伸缩基准值，剩余的空间按比率进行伸缩。具体语法如下：

```
flex-basis：<length> | <percentage> | auto | content
```

取值说明如下：

- ☑ <length>：用长度值来定义宽度。不允许负值。
- ☑ <percentage>：用百分比来定义宽度。不允许负值。
- ☑ auto：无特定宽度值，取决于其他属性值。
- ☑ content：基于内容自动计算宽度。

【补充】

flex 是 flex-grow、flex-shrink 和 flex-basis 三个属性的复合属性，该属性适用于伸缩项目。其中第二个和第三个参数（flex-shrink、flex-basis）是可选参数。默认值为“0 1 auto”。具体语法如下：

```
flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
```

5．对齐方式

align-self 用来在单独的伸缩项目上覆写默认的对齐方式。具体语法如下：

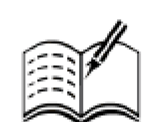
```
align-self：auto | flex-start | flex-end | center | baseline | stretch
```

属性值与 align-items 的属性值相同。

【示例 1】以上面示例为基础，定义伸缩项目在当前位置向右错移一个位置，其中第一个项目位于第二项目的位置，第二个项目位于第三个项目的位置，最后一个项目移到第一个项目的位置，演示效果如图 23.18 所示。

```
<style type="text/css">
.flex-container {
    display: -webkit-flex;
    display: flex;
    width: 500px; height: 300px;border: solid 1px red;
}
.flex-item { background-color: blue; width: 200px; height: 200px; margin: 10px;}
.flex-item:nth-child(0){
    -webkit-order: 4;
    order: 4;
}
.flex-item:nth-child(1){
    -webkit-order: 1;
```






```
        order: 1;
    }
    .flex-item:nth-child(2){
        -webkit-order: 2;
        order: 2;
    }
    .flex-item:nth-child(3){
        -webkit-order: 3;
        order: 3;
    }
    </style>
```

【示例 2】“margin: auto;”在伸缩盒中具有强大的功能，一个“auto”的 margin 会合并剩余的空间。它可以用来把伸缩项目挤到其他位置。下面示例利用“margin-right: auto;”，定义包含的项目居中显示，效果如图 23.19 所示。

```
<style type="text/css">
.flex-container {
    display: -webkit-flex;
    display: flex;
    width: 500px; height: 300px; border: solid 1px red;
}
.flex-item {
    background-color: blue; width: 200px; height: 200px;
    margin: auto;
}
</style>
<div class="flex-container">
    <div class="flex-item">伸缩项目</div>
</div>
```

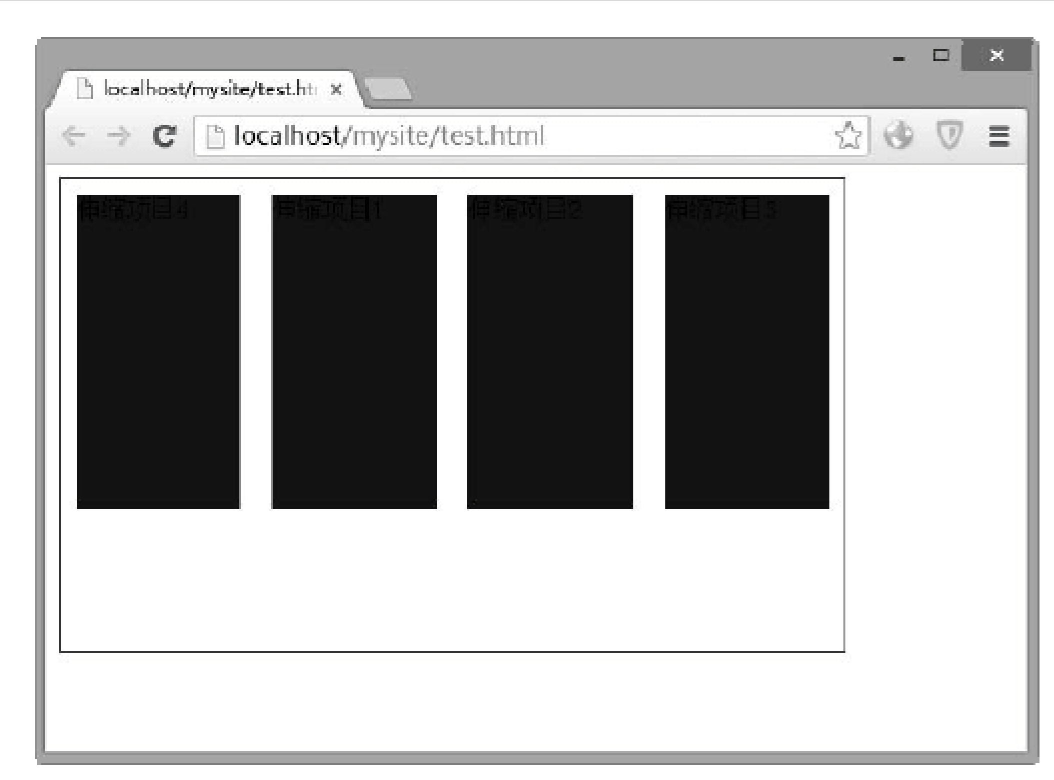


图 23.18　定义伸缩项目错位显示

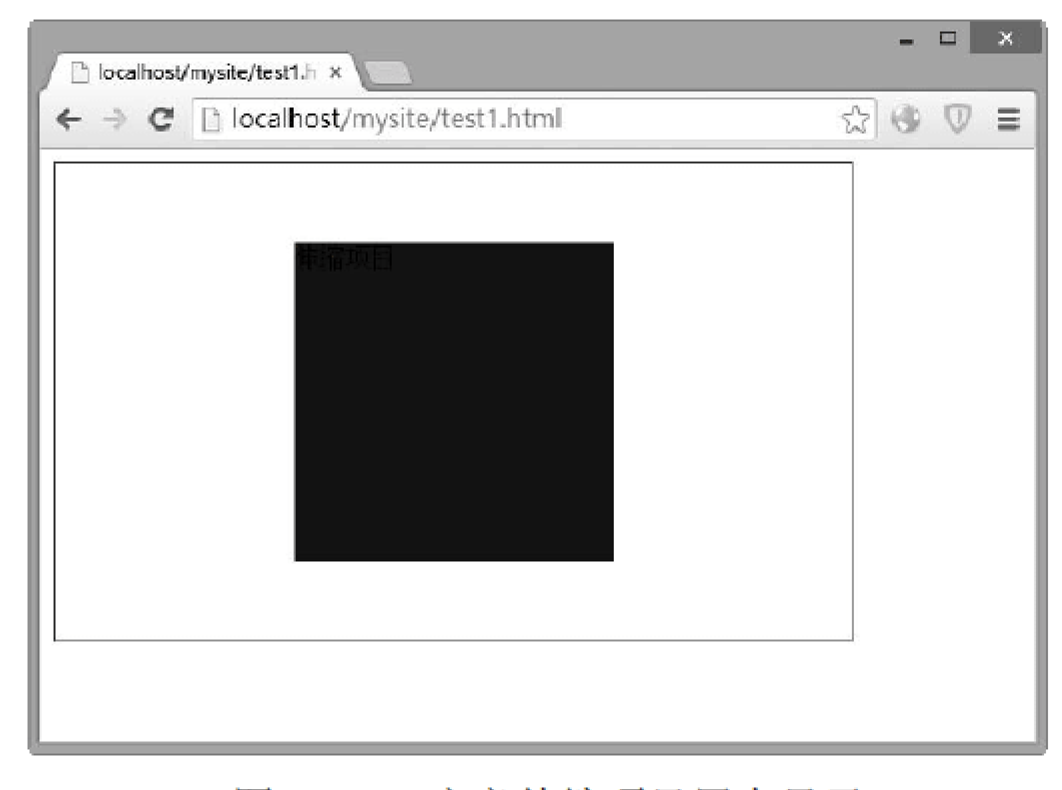
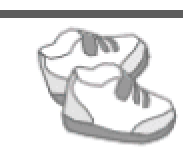


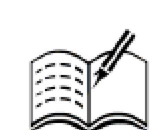
图 23.19　定义伸缩项目居中显示

23.4　伸缩盒版本比较和兼容

本节将重点介绍伸缩盒模型的旧版本、混合版本和新版本之间的用法差异，以及兼容方法，以便用户设计在不同浏览器上都能正确显示的弹性布局。



23.4.1　版本比较和兼容方法



Note

CSS3 伸缩盒布局在不断发展中，并不断升级，大致经历了三个阶段，并逐步达到稳定，主流浏览器对新版本也开始完整的支持。

☑　2009 年版本（旧版本）：“display:box;”。

☑　2011 年版本（混合版本）：“display:flexbox;”。

☑　2012 年版本（新版本）：“display:flex;”。

如果把 Flexbox 新语法、旧语法和混合语法混合在一起使用，就可以让浏览器得到完美的展示。当然，在使用 Flexbox 时，应该考虑不同浏览器的私有属性，如 Chrome 浏览器要添加前缀-webkit-，Firefox 浏览器要添加前缀-moz-等。

1．浏览器支持状况

各主流浏览器对 Flexbox 规范不同版本的支持如表 23.1 所示。

表 23.1　浏览器对规范版本的支持

规 范 版 本	IE	Opera	Firefox	Chrome	Safari
新版本（标准版）	11	12.10+ *	22	29+、21–28 (-webkit-)	
混合版本	10 (-ms-)				
旧版本			3–21 (-moz-)	<21 (-webkit-)	3–6 (-webkit-)

2．开启 Flexbox

不同 Flexbox 版本定义一个元素为伸缩容器的方法比较如表 23.2 所示。

表 23.2　比较启动 Flexbox

规 范 版 本	属 性 名 称	块伸缩容器	内联伸缩容器
新版本（标准版）	display	flex	inline-flex
混合版本	display	flexbox	inline-flexbox
旧版本	display	box	inline-box

3．主轴对齐方式

不同 Flexbox 版本指定伸缩项目沿主轴对齐方式的取值比较如表 23.3 所示。

表 23.3　比较主轴对齐方式

规 范 版 本	属 性 名 称	start	center	end	justify	distribute
新版本（标准版）	justify-content	flex-start	center	flex-end	space-between	space-around
混合版本	flex-pack	start	center	end	justify	distribute
旧版本	box-pack	start	center	end	justify	N/A

提示：☑　start：开始位置。

☑　center：中间位置。

☑　end：结束位置。

☑　justify：两端对齐。

☑　distribute：均匀对齐。

☑　N/A：表示不适用的意思。



Note

4．侧轴对齐方式

不同 Flexbox 版本指定伸缩项目沿侧轴对齐方式的取值比较如表 23.4 所示。

表 23.4 比较侧轴对齐方式

规 范 版 本	属 性 名 称	start	center	end	baseline	stretch
新版本（标准版）	align-items	flex-start	center	flex-end	baseline	stretch
混合版本	flex-align	start	center	end	baseline	stretch
旧版本	box-align	start	center	end	baseline	stretch

提示：☑ baseline：基线对齐。
☑ stretch：伸展对齐。

5．单个伸缩项目侧轴对齐方式

不同 Flexbox 版本指定单个伸缩项目沿侧轴对齐方式的取值比较如表 23.5 所示。

表 23.5 比较单个伸缩项目侧轴对齐方式

规 范 版 本	属 性 名 称	auto	start	center	end	baseline	stretch
新版本（标准版）	align-self	auto	flex-start	center	flex-end	baseline	stretch
混合版本	flex-item-align	auto	start	center	end	baseline	stretch
旧版本	N/A						

6．伸缩项目行对齐方式

不同 Flexbox 版本指定伸缩项目行在侧轴的对齐方式的取值比较如表 23.6 所示。

表 23.6 比较伸缩项目行对齐方式

规 范 版 本	属 性 名 称	start	center	end	justify	distribute	stretch
新版本（标准版）	align-content	flex-start	center	flex-end	space-between	space-around	stretch
混合版本	flex-line-pack	start	center	end	justify	distribute	stretch
旧版本	N/A						

注意：只有伸缩项目有多行时才生效，这种情况发生在只有伸缩容器设置了 flex-wrap 为 wrap 时，并且没有足够的空间把伸缩项目放在同一行中。这个将对每一行起作用而不是每一个伸缩项目。

7．显示顺序

不同 Flexbox 版本指定伸缩项目的显示顺序的取值比较如表 23.7 所示。

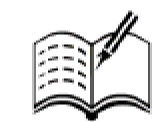
表 23.7 比较显示顺序

规 范 版 本	属 性 名 称	属 性 值
新版本（标准版）	order	<number>
混合版本	flex-order	<number>
旧版本	box-ordinal-group	<integer>



8．伸缩性

不同 Flexbox 版本指定伸缩项目如何伸缩，比较如表 23.8 所示。



Note

表 23.8　比较伸缩性

规 范 版 本	属 性 名 称	属 性 值
新版本（标准版）	flex	none | [<flex-grow> <flex-shrink>? || <flex-basis>]
混合版本	flex	none | [[<pos-flex> <neg-flex>?] || <preferred-size>]
旧版本	box-flex	<number>

flex 属性在微软的草案与新标准或多或少不一样。主要区别在于：它们都转换成标准缩写版本，属性值为 flex-grow、flex-shrink 和 flex-basis，值使用相同的方式在速记。然而，flex-shrink（以前称为负 flex）的默认值为 1。这意味着伸缩项目默认不能收缩。以前，空间不足使用 flex-shrink 比例来收缩伸缩项目，但现在可以在 flex-basis 的基础上配合 flex-shrink 来收缩伸缩项目。

9．伸缩流

不同 Flexbox 版本指定伸缩容器主轴的伸缩流方向比较如表 23.9 所示。

表 23.9　比较伸缩流

规 范 版 本	属 性 名 称	Horizontal	Reversed horizontal	Vertical	Reversed vertical
新版本（标准版）	flex-direction	row	row-reverse	column	column-reverse
混合版本	flex-direction	row	row-reverse	column	column-reverse
旧版本	box-orient box-direction	Horizontal normal	Horizontal reverse	Vertical normal	Vertical reverse

在旧版本规范中，使用 box-direction 属性设置为 reverse 和在新版本中设置 row-reverse 或 column-reverse 得到的效果相同。如果想要的效果是 row 或 column，可以省略不设置，因为 normal 是默认的初始值。

当设置 direction 为 reverse，主轴就翻转。例如，当使用“ltr”书写模式，指定 row-reverse 时，所有伸缩项目会从右向左排列。类似地，column-reverse 将会使所有伸缩项目从下向上排列，来代替从上往下排列。

在旧版本中，需要使用 box-orient 来设置书写模式的方向。当使用“ltr”模式时，horizontal 可用在 inline-axis，vertical 可用在 block-axis。如果使用的是一个自上而下的书写模式，如东亚传统的书写模式，这些值就会翻转。

10．换行

不同 Flexbox 版本指定伸缩项目是否沿着侧轴排列，比较如表 23.10 所示。

表 23.10　比较换行

规 范 版 本	属 性 名 称	No wrapping	Wrapping	Reversed wrap
新版本（标准版）	flex-wrap	nowrap	wrap	wrap-reverse
混合版本	flex-wrap	nowrap	wrap	wrap-reverse
旧版本	box-lines	single	multiple	N/A



wrap-reverse 让伸缩项目在侧轴上进行 start 和 end 翻转，所以，如果伸缩项目在水平排列，伸缩项目翻转不会到一个新的线下面，它会翻转到一个新的线上面。简单理解就是伸缩项目只是上下或前后翻转顺序。



23.4.2　案例：设计 3 栏页面

下面示例根据 23.2.2 节介绍的方法，使用不同版本语法，设计一个兼容不同设备和浏览器的弹性页面，演示效果如图 23.20 所示。

具体操作步骤请扫码学习。





线上阅读　视频讲解



图 23.20　定义混合伸缩盒布局

23.4.3　案例：设计 3 行 3 列应用

本例借助 Flexbox 伸缩盒布局，设计页面呈现 3 行 3 列布局样式，同时能够根据窗口自适应调整各自空间，以满屏显示，效果如图 23.21 所示。

具体代码解析请扫码学习。





线上阅读　视频讲解

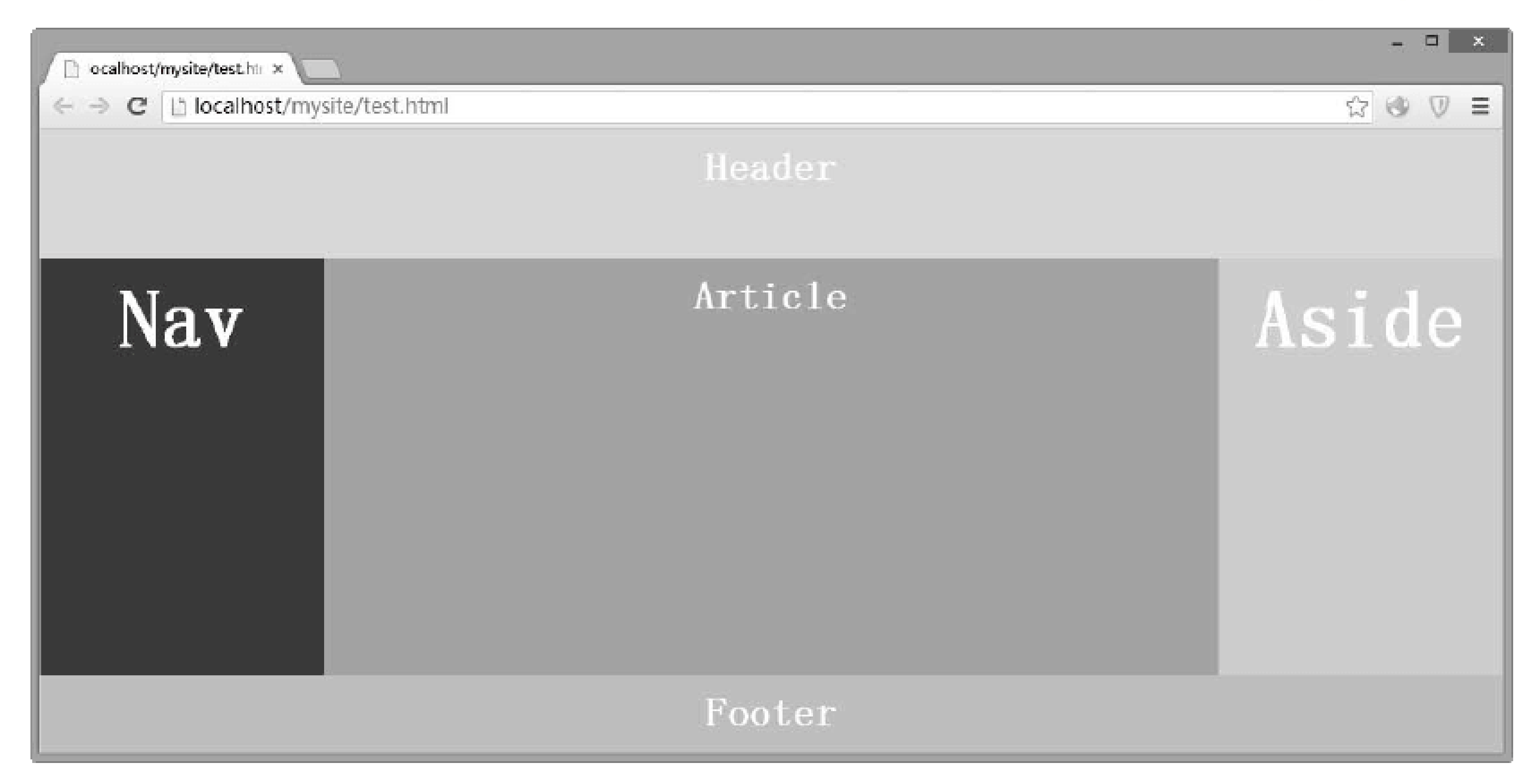
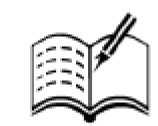


图 23.21　HTML5 应用文档



23.5 在线练习



Note

Flexbox 3 个不同版本的规范对应着不同的实现。需要关注哪个版本，取决于需要支持的浏览器。



在线练习

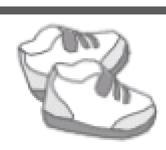
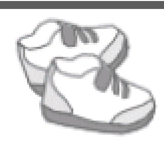
第 24 章

CSS3 动画

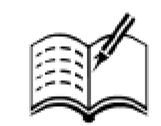
CSS3 动画包括过渡动画和关键帧动画，它们主要通过改变 CSS 属性值来模拟实现。本章将详细介绍 Transform、Transitions 和 Animations 三大功能模块，其中 Transform 实现对网页对象的变形操作，Transitions 实现 CSS 属性过渡变化，Animations 实现 CSS 样式分步式演示效果。

【学习重点】

- 设计对象变形操作。
- 设计过渡样式。
- 设计关键帧动画。
- 能够灵活使用 CSS3 动画设计页面特效。

24.1 CSS3 变形


Note


权威参考

2012 年 9 月，W3C 发布了 CSS3 变形工作草案。CSS3 变形允许 CSS 把元素转变为 2D 或 3D 空间，这个草案包括了 CSS3 2D 变形和 CSS3 3D 变形。

权威参考：http://www.w3.org/TR/css-transforms-1/。

24.1.1 认识 Transform

CSS3 变形是多种效果的集合，如旋转、缩放、平移和倾斜等，每个效果都被称为变形函数，它们可以操控元素发生旋转、缩放、平移和倾斜等变化。在 CSS3 之前，实现类似的效果都需要图片、Flash 或 JavaScript 才能完成。而使用纯 CSS 来完成这些变形则无须加载这些额外的文件，提升了开发效率，提高了页面的执行效率。

CSS3 变形包括 3D 变形和 2D 变形，3D 变形使用基于 2D 变形的相同属性，如果了解了 2D 变形，会发现 3D 变形与 2D 变形的功能类似。

CSS 2D Transform 获得了各主流浏览器的支持，但是 CSS 3D Transform 支持程度不是很完善。考虑到浏览器兼容性，3D 变形在实际应用时应添加私有属性，并且个别属性在某些主流浏览器中并未得到很好的支持，简单说明如下：

☑ 在 IE 10+中，3D 变形部分属性未得到很好的支持。

☑ Firefox 10.0 至 Firefox 15.0 版本的浏览器，在使用 3D 变形时需要添加私有属性-moz-，但从 Firefox 16.0+版本开始无须添加浏览器私有属性。

☑ Chrome 12.0+版本中使用 3D 变形时需要添加私有属性-webkit-。

☑ Safari 4.0+版本中使用 3D 变形时需要添加私有属性-webkit-。

☑ Opera 15.0+版本才开始支持 3D 变形，使用时需要添加私有属性-webkit-。

☑ 移动设备中 iOS Safari 3.2+、Android Browser 3.0+、Blackberry Browser 7.0+、Opera Mobile 24.0+、Chrome for Android 25.0+都支持 3D 变形，但在使用时需要添加私有属性-webkit-；Firefox for Android 19.0+支持 3D 变形，但无须添加浏览器私有属性。


视频讲解

24.1.2 设置原点

CSS 变形的原点默认为对象的中心点（50% 50%），使用 transform-origin 属性可以重新设置新的变形原点。语法格式如下所示：

```
transform-origin:[ <percentage> | <length> | left | center① | right ] [ <percentage> | <length> | top | center② | bottom ]?
```

取值简单说明如下：

☑ <percentage>：用百分比指定坐标值。可以为负值。

☑ <length>：用长度值指定坐标值。可以为负值。

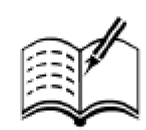
☑ left：指定原点的横坐标为 left。

☑ center①：指定原点的横坐标为 center。

☑ right：指定原点的横坐标为 right。

☑ top：指定原点的纵坐标为 top。





Note

☑　center②：指定原点的纵坐标为 center。

☑　bottom：指定原点的纵坐标为 bottom。

【示例】通过重置变形原点，可以设计不同的变形效果。在下面示例中以图像的右上角为原点逆时针旋转图像 45°，则比较效果如图 24.1 所示。

```
<style type="text/css">
img {/*  固定两幅图像相同大小和相同显示位置  */
    position: absolute;
    left: 20px;
    top: 10px;
    width: 170px;
    width: 250px;
}
img.bg {/*  设置第 1 幅图像作为参考  */
    opacity: 0.3;
    border: dashed 1px red;
}
img.change {/*  变形第 2 幅图像  */
    border: solid 1px red;
    transform-origin: top right;     /*以右上角为原点进行变形*/
    transform: rotate(-45deg);       /*逆时针旋转 45 度*/
}
</style>
<img class="bg" src="images/1.jpg">
<img class="change" src="images/1.jpg">
```

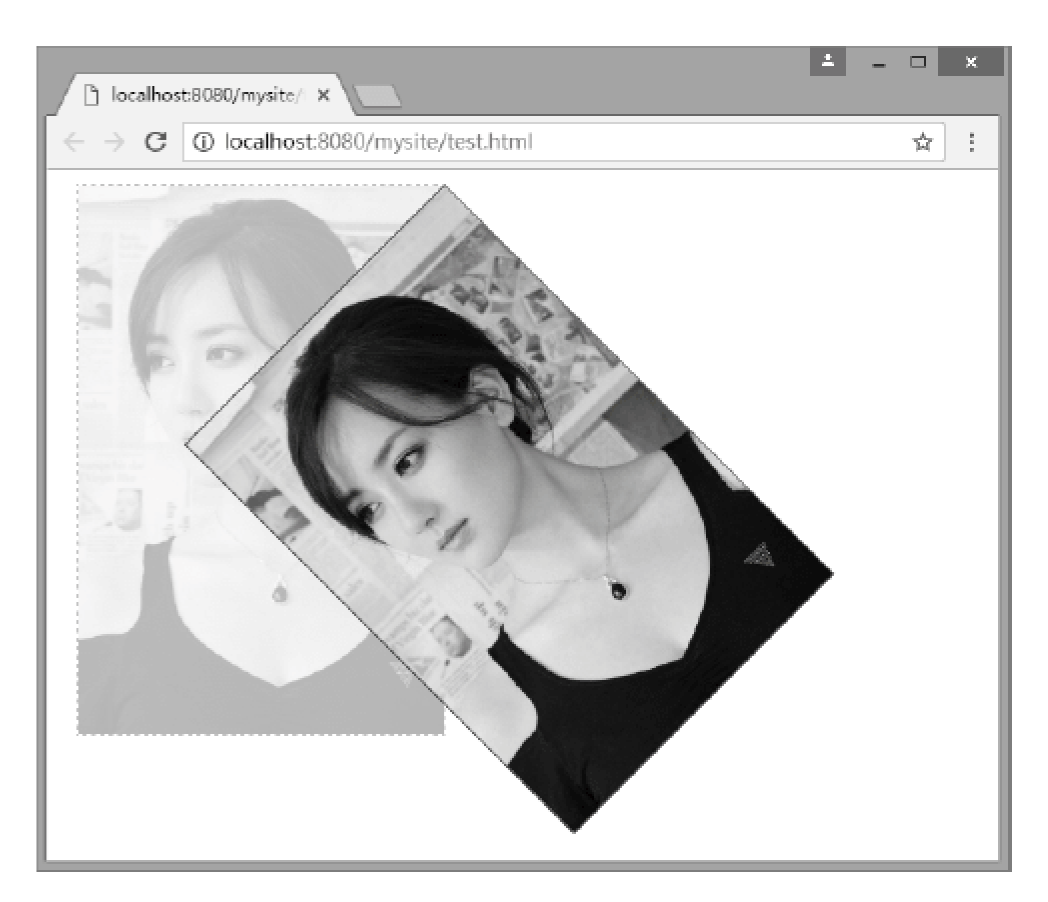


图 24.1　自定义旋转原点

24.1.3　2D 旋转



视频讲解

rotate()函数能够在 2D 空间内旋转对象，语法格式如下：

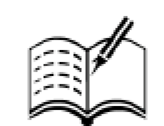
```
rotate(<angle>)
```

参数 angle 表示角度值，取值单位可以是：度，如 90deg（90°，一圈为 360°）；梯度，如 100grad（相当于 90°，360°等于 400grad）；弧度，如 1.57rad（约等于 90°，360°等于 2π）；圈，如 0.25turn

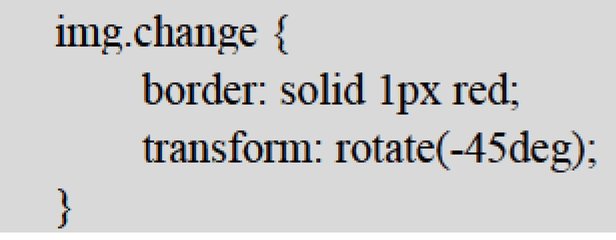


（等于 90°，360°等于 1turn）。

【示例】以 24.1.2 节示例为基础，下面按默认原点逆时针旋转图像 45°，效果如图 24.2 所示。



Note



```
img.change {
    border: solid 1px red;
    transform: rotate(-45deg);
}
```

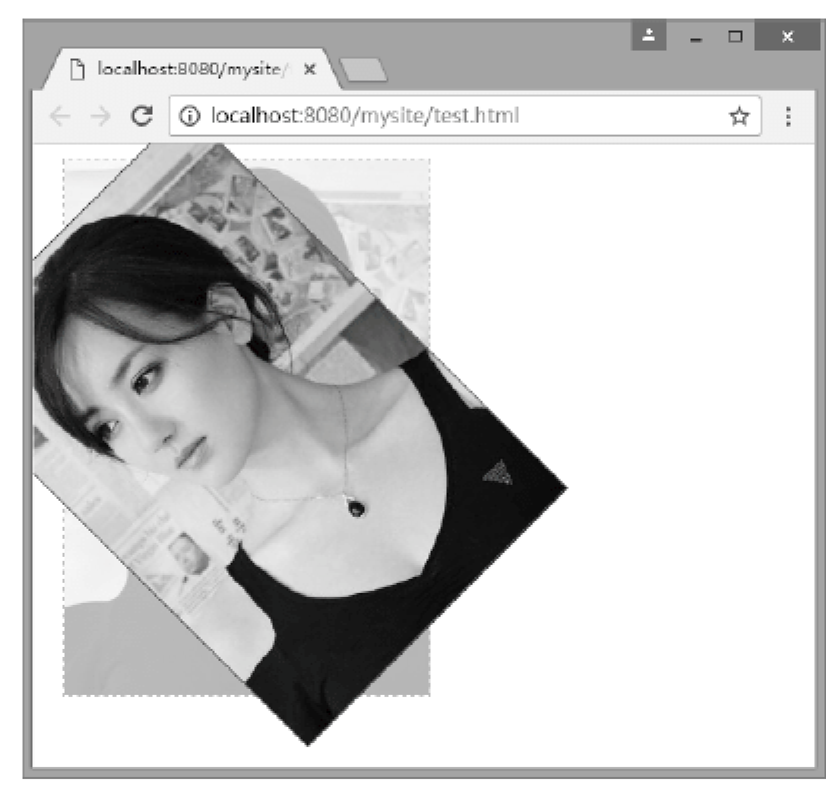


图 24.2　定义旋转效果



视 频 讲 解

24.1.4　2D 缩放

scale()函数能够缩放对象大小，语法格式如下：

```
scale(<number>[, <number>])
```

该函数包含两个参数值，分别用来定义宽和高缩放比例。取值简单说明如下：

- ☑ 如果取值为正数，则基于指定的宽度和高度将放大或缩小对象。
- ☑ 如果取值为负数，则不会缩小元素，而是翻转元素（如文字被反转），然后再缩放元素。
- ☑ 如果取值为小于 1 的小数（如 0.5），可以缩小元素。
- ☑ 如果第二个参数省略，则第二个参数等于第一个参数值。

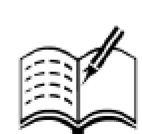
【示例】继续以 24.1.2 节示例为基础，下面按默认原点把图像缩小一倍，效果如图 24.3 所示。



图 24.3　缩小对象一倍效果



```
img.change {
    border: solid 1px red;
    transform: scale(0.5);
}
```





视频讲解

24.1.5 2D 平移

translate()函数能够平移对象的位置，语法格式如下：

```
translate(<translation-value>[, <translation-value>])
```

该函数包含两个参数值，分别用来定义对象在 X 轴和 Y 轴相对于原点的偏移距离。如果省略参数，则默认值为 0。如果取负值，则表示反向偏移，参考原点保持不变。

【示例】下面示例设计向右下角方向平移图像，其中 X 轴偏移 150 像素，Y 轴偏移 50 像素，演示效果如图 24.4 所示。

```
img.change {
    border: solid 1px red;
    transform: translate(150px, 50px);
}
```

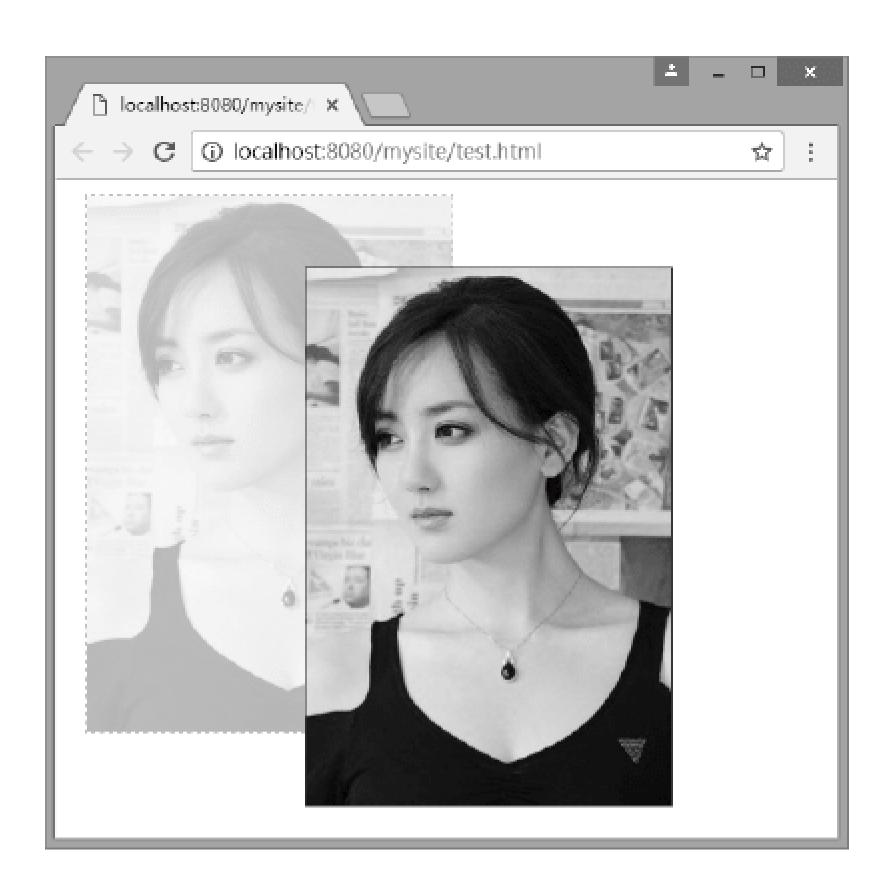


图 24.4 平移对象效果



视频讲解

24.1.6 2D 倾斜

skew()函数能够倾斜显示对象，语法格式如下：

```
skew(<angle> [, <angle>])
```

该函数包含两个参数值，分别用来定义对象在 X 轴和 Y 轴倾斜的角度。如果省略参数，则默认值为 0。与 rotate()函数不同，rotate()函数只是旋转对象的角度，而不会改变对象的形状；skew()函数会改变对象的形状。

【示例】下面示例使用 skew()函数变形图像，X 轴倾斜 30°，Y 轴倾斜 20°，效果如图 24.5 所示。

```
img.change {
    border: solid 1px red;
```



```
        transform: skew(30deg, 20deg);
    }
```



Note

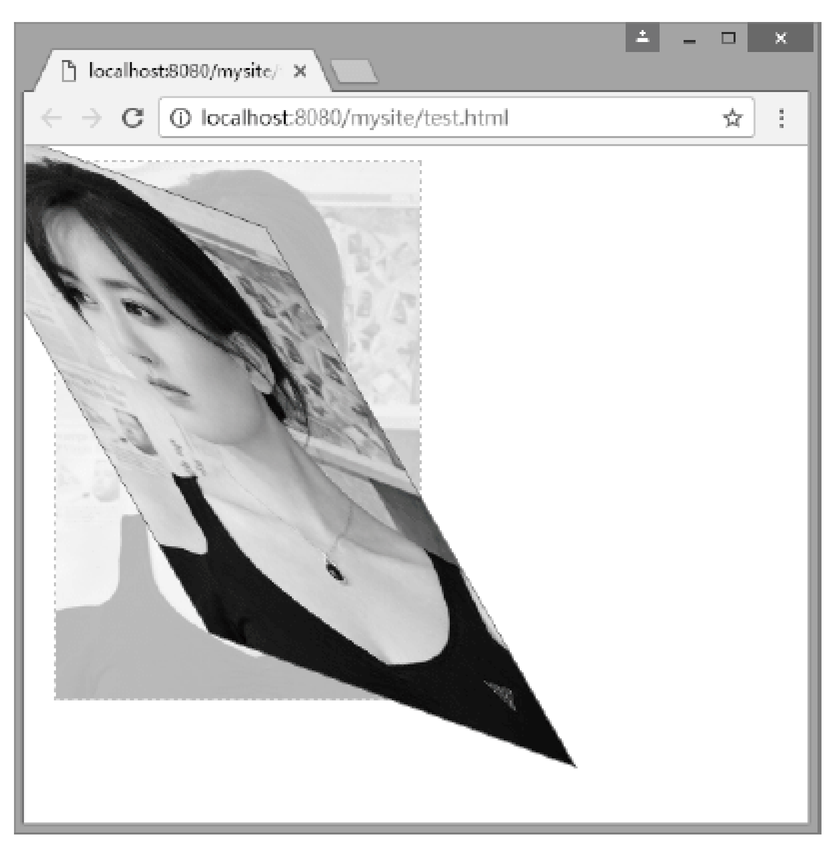


图 24.5　倾斜对象效果



视 频 讲 解

24.1.7　2D 矩阵

matrix()是一个矩阵函数，它可以同时实现缩放、旋转、平移和倾斜操作，语法格式如下：

```
matrix(<number>, <number>, <number>, <number>, <number>, <number>)
```

该函数包含 6 个值，具体说明如下：

☑　第 1 个参数控制 X 轴缩放；

☑　第 2 个参数控制 X 轴倾斜；

☑　第 3 个参数控制 Y 轴倾斜；

☑　第 4 个参数控制 Y 轴缩放；

☑　第 5 个参数控制 X 轴平移；

☑　第 6 个参数控制 Y 轴平移。

【示例】下面示例使用 matrix()函数模拟 24.1.6 节示例的倾斜变形操作，效果类似 24.1.6 节示例效果。

```
img.change {
    border: solid 1px red;
    transform: matrix(1, 0.6, 0.2, 1, 0, 0);
}
```

【补充】

多个变形函数可以在一个声明中同时定义。例如：

```
div {
    transform: translate(80, 80);
    transform: rotate(45deg);
    transform: scale(1.5, 1.5);
}
```

针对上面样式，可以简化为：

```
div { transform: translate(80, 80) rotate(45deg) scale(1.5, 1.5);}
```







24.1.8　设置变形类型

CSS3 变形包括 2D 和 3D 两种类型，使用 transform-style 属性可以设置 CSS 变形的类型，语法格式如下：

```
transform-style:flat | preserve-3d
```

取值简单说明如下：

☑　flat：指定子元素位于该元素所在平面内进行变形，即 2D 平面变形，为默认值。

☑　preserve-3d：指定子元素定位在三维空间内进行变形，即 3D 立体变形。

【示例】借助上面示例，下面示例使用<div id="box">容器包含两幅图像，改进后的 HTML 结构如下所示。

```
<div id="box">
    <img class="bg" src="images/1.jpg">
    <img class="change" src="images/1.jpg">
</div>
```

为<div id="box">容器设置 CSS3 变形类型为 3D，样式代码如下：

```
#box {
    transform-style: preserve-3d;
}
```

为 change 图像应用 3D 顺时针旋转 45°，CSS 样式如下：

```
img.change {
    border: solid 1px red;
    transform: translate3d(60px, 60px, 400px);
}
```

在浏览器中预览，如图 24.6 所示。

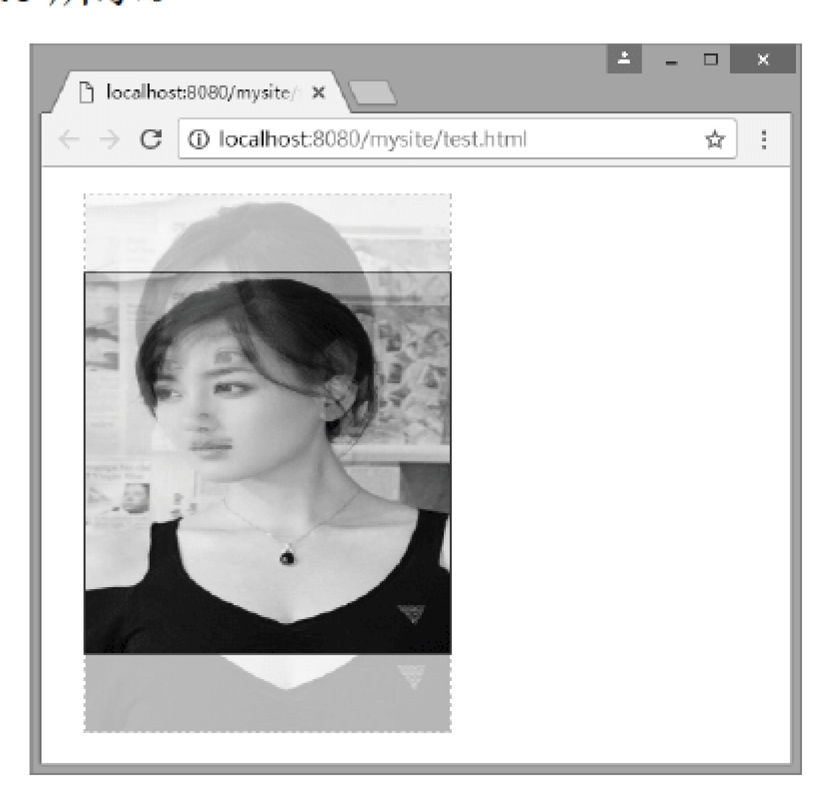


图 24.6　3D 平移效果

24.1.9　设置透视距离和原点



3D 变形与 2D 变形最大的不同就在于其参考的坐标轴不同：2D 变形的坐标轴是平面的，只存在 X 轴和 Y 轴，而 3D 变形的坐标轴则是 x、y、z 三条轴组成的立体空间，X 轴正向、Y 轴正向、Z 轴





Note

正向分别朝向右、下和屏幕外，示意图如图 24.7 所示。

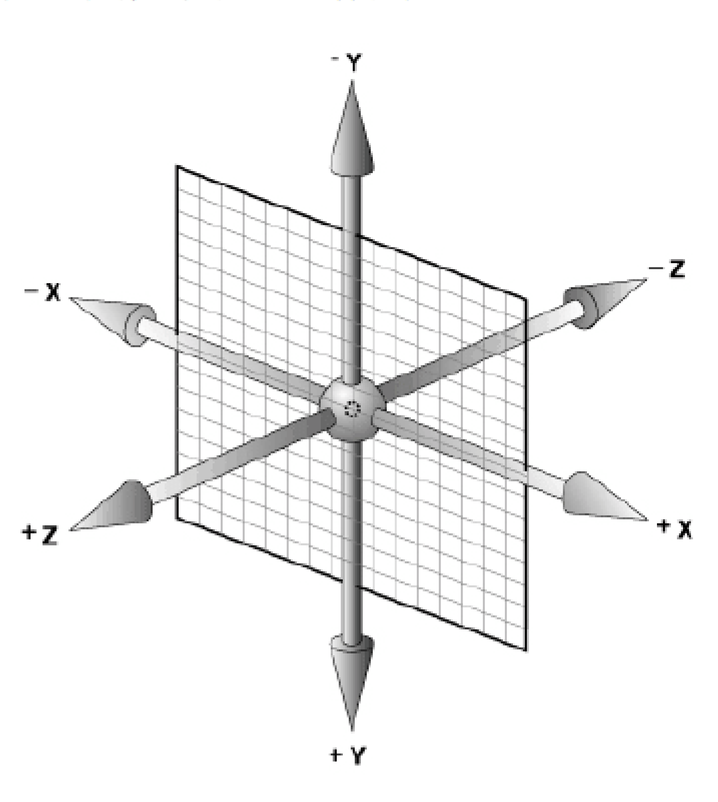


图 24.7　3D 坐标轴示意图

透视是 3D 变形中最重要的概念。如果不设置透视，元素的 3D 变形效果将无法实现。在上节示例中，使用函数 rotateX(45deg)将图像以 X 轴方向为轴沿顺时针旋转 45°，由于没有设置透视样式的效果，可以看到浏览器将图像的 3D 变形操作垂直投射到 2D 视平面上，最终呈现出来的只是图像的宽高变化。

【示例 1】如果在 24.1.8 节示例基础上，在<div id="box">容器外，设置透视点距离为 1200 像素，样式代码如下：

```
body{
    perspective: 1200px;
}
```

在浏览器中可以看到如图 24.8 所示的变形效果。

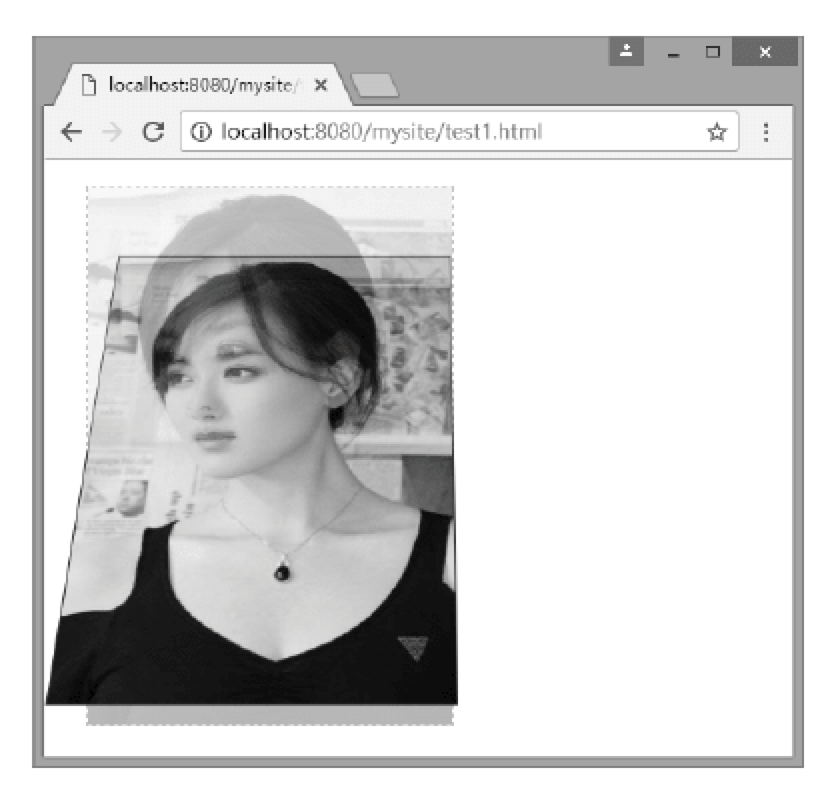


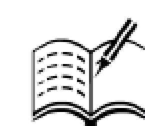
图 24.8　沿 X 轴 3D 旋转 45°效果图

基于对上面示例的直观体验，下面来了解几个核心概念：变形元素、观察者和被观察元素，关系如图 24.9 所示。

☑　变形元素：就是需要进行 3D 变形的元素。主要设置 transform、transform-origin、backface-visibility 等属性。

☑　观察者：就是浏览器模拟出来的用来观察被观察元素的一个没有尺寸的点，观察者发出视线，





类似于一个点光源发出光线。

☑　被观察元素：也称被透视元素，就是被观察者观察的元素，根据属性设置的不同，它有可能是变形对象本身，也可能是它的父级或祖先元素，主要设置 perspective、perspective-origin 等属性。

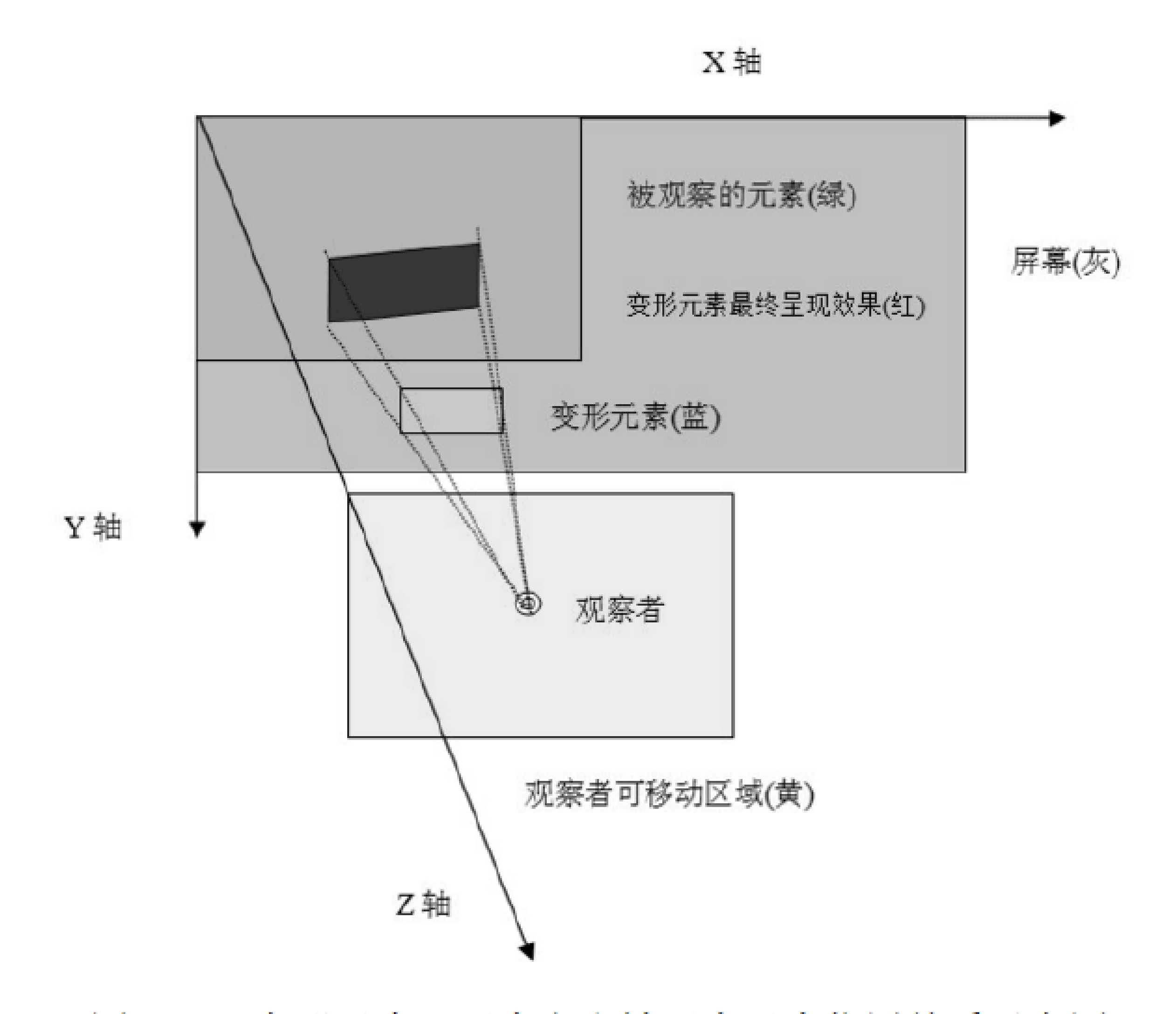


图 24.9　变形元素、观察者和被观察元素位置关系示意图

1．透视距离

透视距离是指观察者沿着平行于 Z 轴的视线与屏幕之间的距离，也称为视距，示意图如图 24.10 所示。

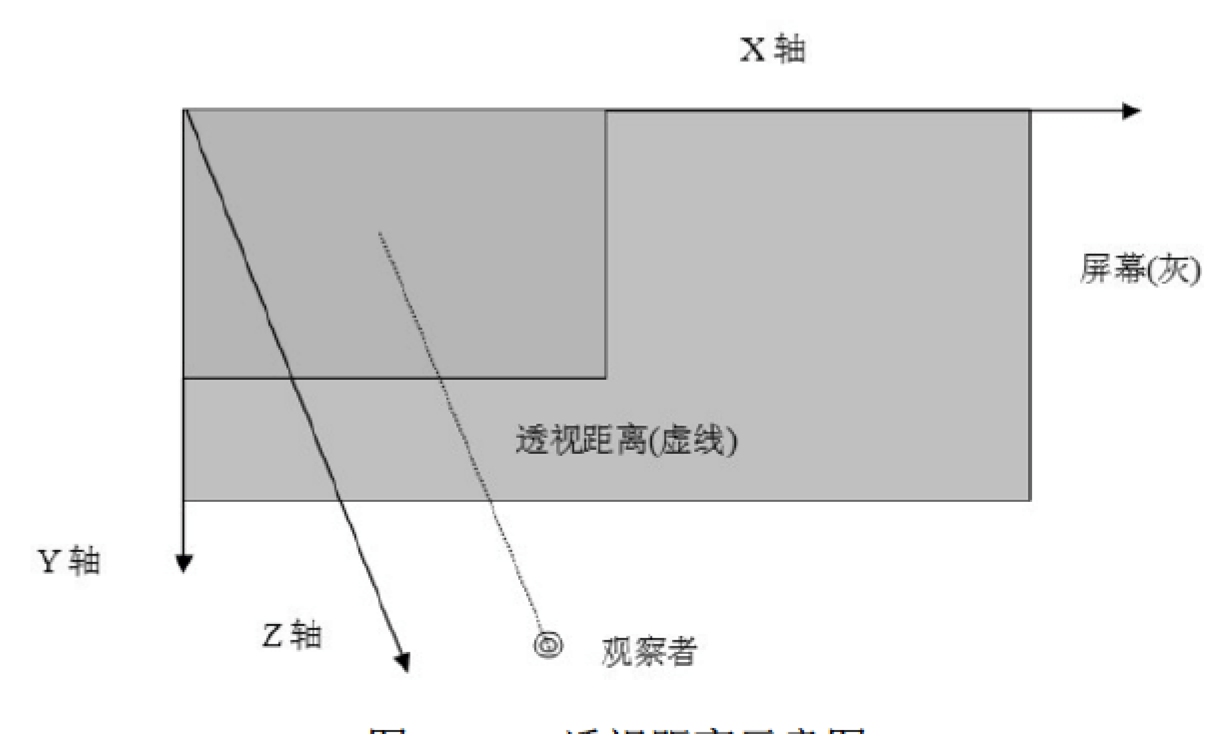


图 24.10　透视距离示意图

使用 perspective 属性可以定义透视距离，语法格式如下：

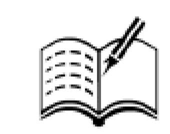
```
Perspective:none | <length>
```

取值简单说明如下：

☑　none：不指定透视。

☑　<length>：指定观察者距离平面的距离，为元素及其内容应用透视变换。





Note

注意：透视距离不可为0和负数，因为观察者与屏幕距离为0时或者在屏幕背面时是不可以观察到被透视元素的正面的。perspective 也不可取百分比，因为百分比需要相对的元素，但Z轴并没有可相对的元素尺寸。

一般地，物体离得越远，显得越小。反映在 perspective 属性上，就是该属性值越大，元素的3D变形效果越不明显。

设置 perspective 属性的元素就是被透视元素。一般地，该属性只能设置在变形元素的父级或祖先级。因为浏览器会为其子级的变形产生透视效果，但并不会为其自身产生透视效果。应用示例可以参考上面示例1。

2．透视原点

透视原点是指观察者的位置，一般观察者位于与屏幕平行的另一个平面上，观察者始终是与屏幕垂直的。观察者的活动区域是被观察元素的盒模型区域，示意图如图24.11所示。

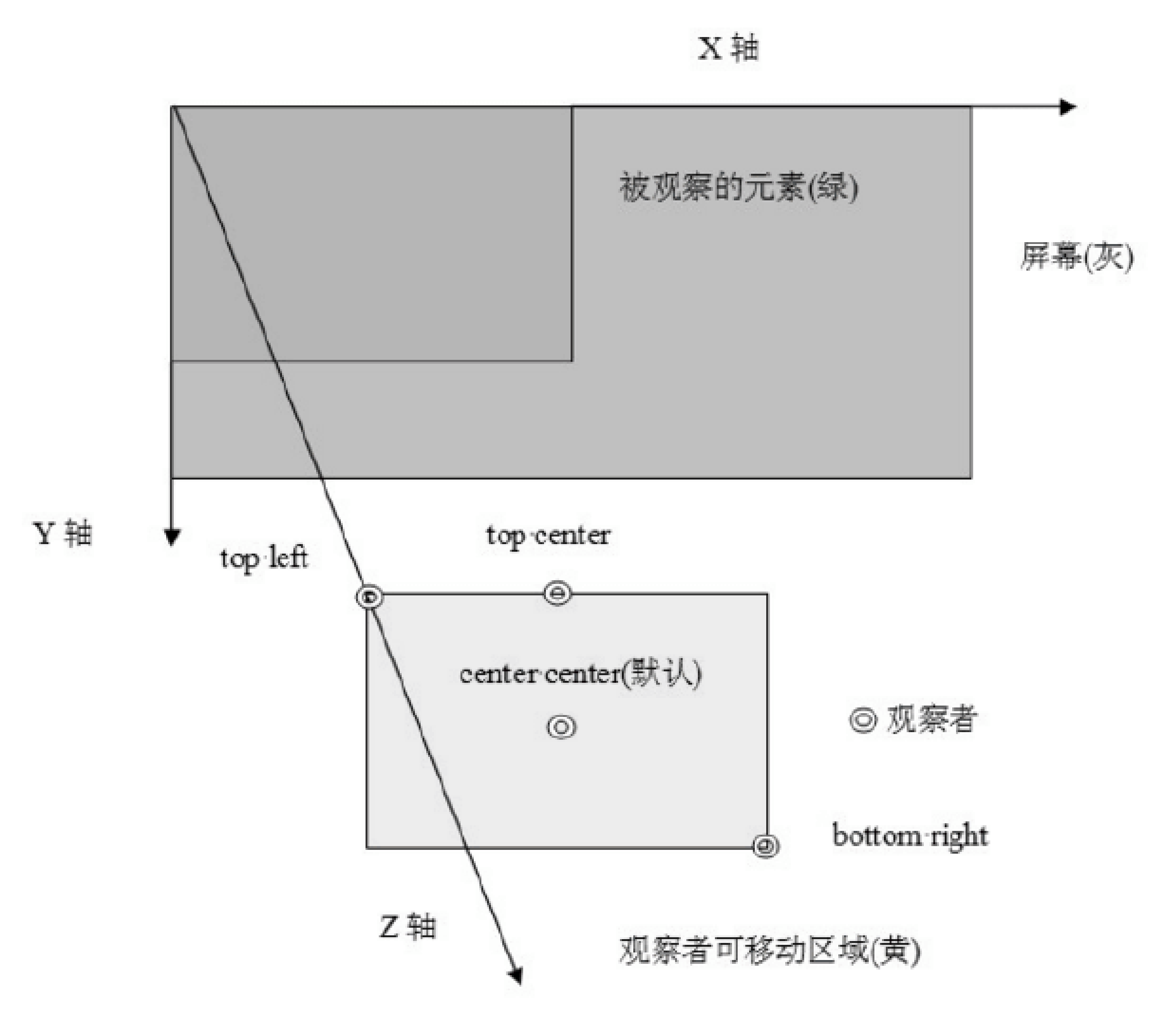


图24.11　下面黄色区域为透视原点的位置区域

使用 perspective-origin 属性可以定义透视点的位置，语法格式如下：

```
perspective-origin:[ <percentage> | <length> | left | center① | right ] [ <percentage> | <length> | top | center② | bottom ]?
```

取值简单说明如下：

☑ <percentage>：用百分比指定透视点坐标值，相对于元素宽度。可以为负值。
☑ <length>：用长度值指定透视点坐标值。可以为负值。
☑ left：指定透视点的横坐标为 left。
☑ center①：指定透视点的横坐标为 center。
☑ right：指定透视点的横坐标为 right。
☑ top：指定透视点的纵坐标为 top。
☑ center②：指定透视点的纵坐标为 center。
☑ bottom：指定透视点的纵坐标为 bottom。





Note

【示例 2】在示例 1 基础上，设置观察点位置在右侧居中位置，则显示效果如图 24.12 所示。

```
body{
    perspective: 1200px;
    perspective-origin: right;
}
```

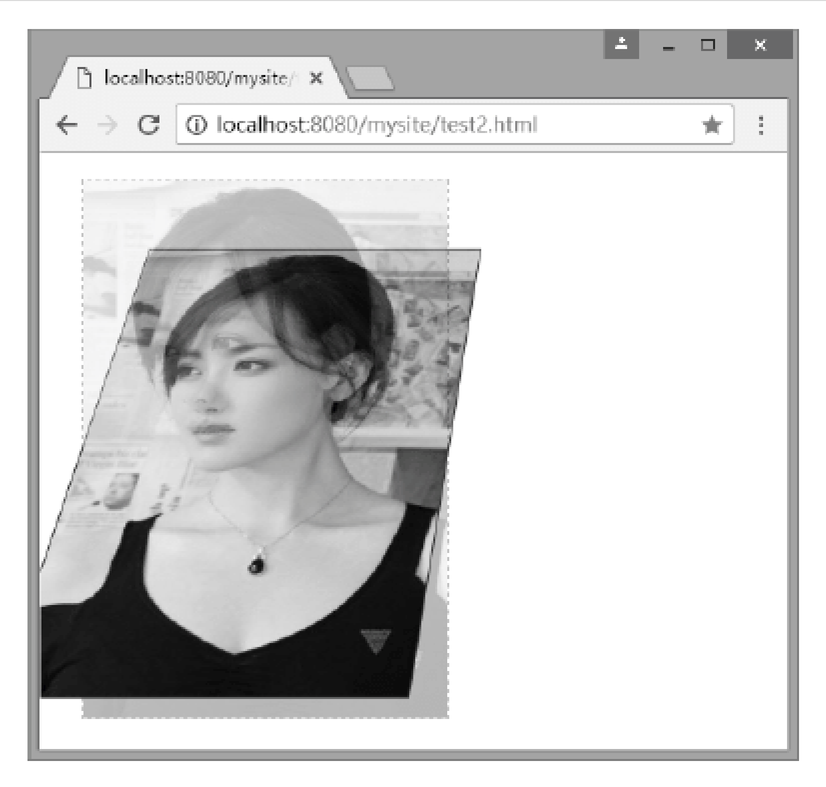


图 24.12　设置观察点位置在右侧的效果

24.1.10　3D 平移



视频讲解

3D 平移主要包括下面 4 个函数：

☑　translatex(<translation-value>)：指定对象 X 轴（水平方向）的平移。

☑　translatey(<translation-value>)：指定对象 Y 轴（垂直方向）的平移。

☑　translatez(<length>)：指定对象 Z 轴的平移。

☑　translate3d(<translation-value>,<translation-value>,<length>)：指定对象的 3D 平移。第 1 个参数对应 X 轴，第 2 个参数对应 Y 轴，第 3 个参数对应 Z 轴，参数不允许省略。

参数<translation-value>表示<length>或<percentage>，即 X 轴和 Y 轴可以取值长度值或百分比，但是 Z 轴只能够设置长度值。

【示例】下面示例设计图像在 3D 空间中平移，设计一种错位效果，如图 24.13 所示。

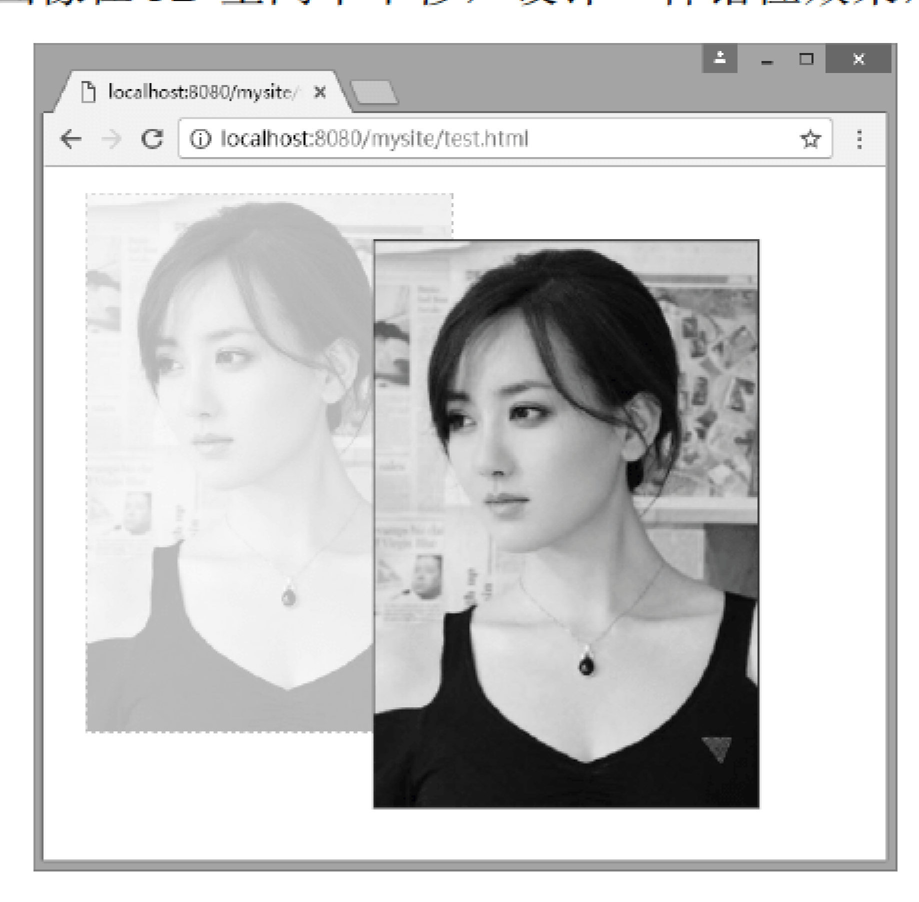


图 24.13　定义 3D 平移效果





Note

```
#box {
    transform-style: preserve-3d;
    perspective: 1200px;
}
img.change {
    border: solid 1px red;
    transform: translate3d(200px, 30px, 60px);
}
```

从图 24.13 效果可以看出，当 Z 轴值越大时，元素离浏览者越近，从视觉上元素就变得更大；反之其值越小时，元素也离观看者越远，从视觉上元素就变得更小。

提示： translateZ()函数在实际使用中等效于 translate3d(0,0,tz)。仅从视觉效果上看，translateZ()和 translate3d(0,0,tz)函数功能非常类似于二维空间的 scale()缩放函数，但实际上完全不同。translateZ()和 translate3d(0,0,tz)变形发生在 Z 轴上，而不是 X 轴和 Y 轴。

视频讲解

24.1.11　3D 缩放

3D 缩放主要包括下面 4 个函数：

- ☑ scalex(<number>)：指定对象 X 轴的（水平方向）缩放。
- ☑ scaley(<number>)：指定对象 Y 轴的（垂直方向）缩放。
- ☑ scalez(<number>)：指定对象的 Z 轴缩放。
- ☑ scale3d(<number>,<number>,<number>)：指定对象的 3D 缩放。第 1 个参数对应 X 轴，第 2 个参数对应 Y 轴，第 3 个参数对应 Z 轴，参数不允许省略。

参数<number>为一个数字，表示缩放倍数，可参考 2D 缩放参数说明。

【示例】 下面以上面示例为基础，在 X 轴和 Y 轴放大图像 1.5 倍，Z 轴放大图像 2 倍，然后使用 translatex()把变形的图像移到右侧显示，以便与原图进行比较，演示效果如图 24.14 所示。

```
img.change {
    border: solid 1px red;
    transform: scale3D(1.5,1.5,2) translatex(240px);
}
```

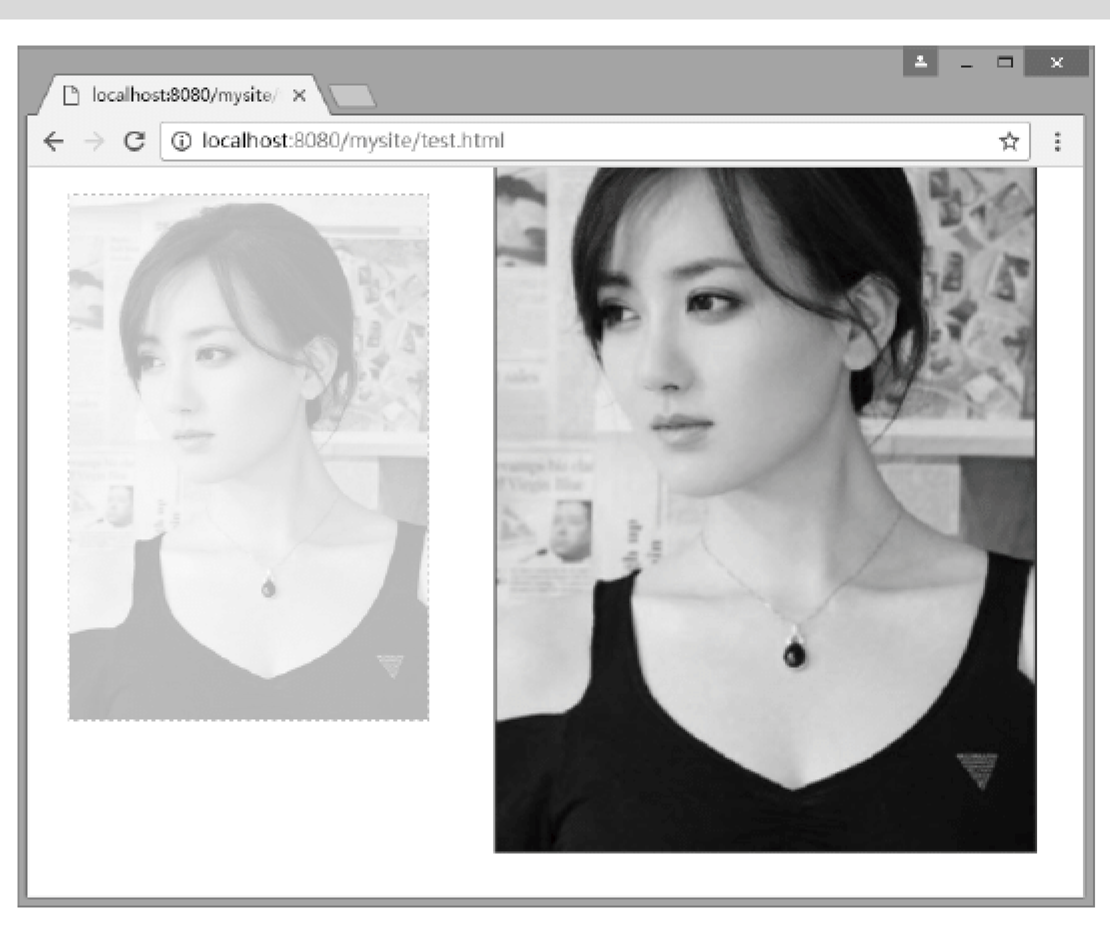


图 24.14　定义 3D 缩放效果





24.1.12　3D 旋转

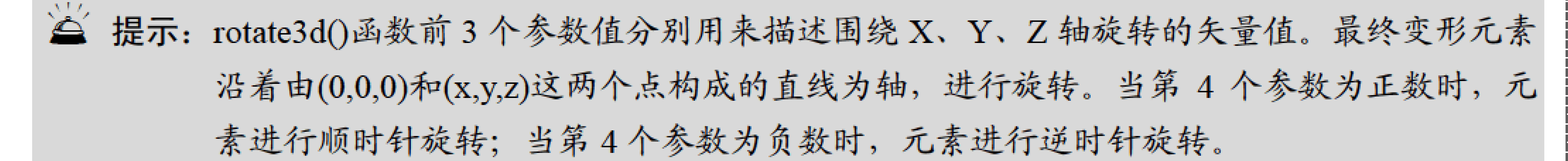
3D 旋转主要包括下面 4 个函数：

☑ rotatex(<angle>)：指定对象在 X 轴上的旋转角度。

☑ rotatey(<angle>)：指定对象在 Y 轴上的旋转角度。

☑ rotatez(<angle>)：指定对象在 Z 轴上的旋转角度。

☑ rotate3d(<number>,<number>,<number>,<angle>)：指定对象的 3D 旋转角度，其中前 3 个参数分别表示旋转的方向 x、y、z，第 4 个参数表示旋转的角度，参数不允许省略。


提示：rotate3d()函数前 3 个参数值分别用来描述围绕 X、Y、Z 轴旋转的矢量值。最终变形元素沿着由(0,0,0)和(x,y,z)这两个点构成的直线为轴，进行旋转。当第 4 个参数为正数时，元素进行顺时针旋转；当第 4 个参数为负数时，元素进行逆时针旋转。

rotate3d()函数可以与前面 3 个旋转函数进行转换，简单说明如下：

☑ rotatex(a)函数功能等同于 rotate3d(1,0,0,a)。

☑ rotatey(a)函数功能等同于 rotate3d(0,1,0,a)。

☑ rotatez(a)函数功能等同于 rotate3d(0,0,1,a)。

【示例】以上面示例为基础，使用 rotate3d()函数顺时针旋转图像 45°，其中 X 轴、Y 轴和 Z 轴比值为 2、2、1，效果如图 24.15 所示。

```
img.change {
    border: solid 1px red;
    transform: rotate3d(2,2,1,45deg);
}
```

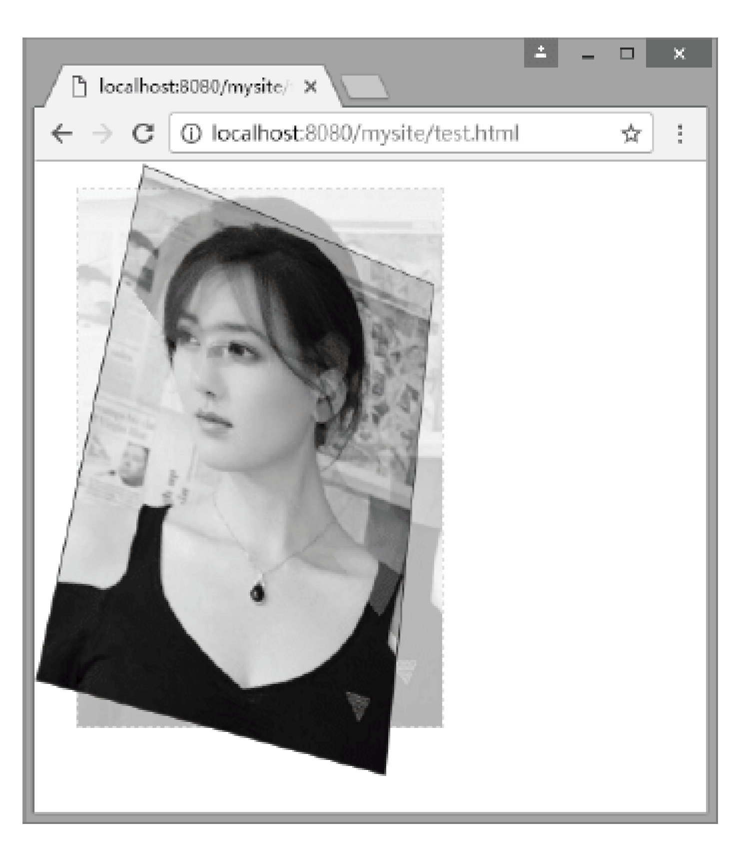


图 24.15　定义 3D 旋转效果

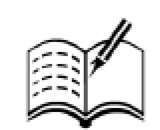
24.1.13　透视函数



perspective 属性可以定义透视距离，它应用在变形元素的父级或祖先级元素上。而透视函数 perspective()是 transform 变形函数的一个属性值，可以应用于变形元素本身。具体语法格式如下：

```
perspective(<length>)
```





Note

参数是一个长度值，值只能是正数。

【示例】下面示例设计图像在 X 轴上旋转 120°，透视距离为 1200 像素，如图 24.16 所示。

```
#box { transform-style: preserve-3d;}
img.change {
    border: solid 1px red;
    transform:perspective(180px) rotateX(120deg);
}
```

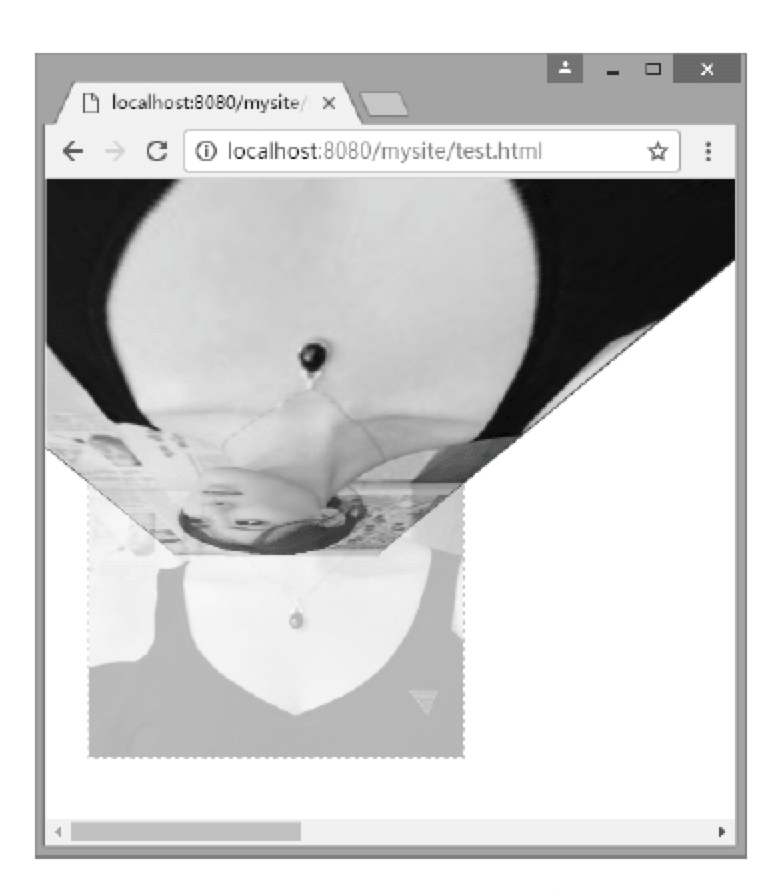


图 24.16　定义 3D 旋转效果

> 注意：由于 transform 属性是以从前向后的顺序解析属性值的，所以一定要把 perspective()函数写在其他变形函数前面，否则将没有透视效果。

由于透视原点 perspective-origin 只能设置在设置了 perspective 透视属性的元素上。若为元素设置透视函数 perspective()，则透视原点不起作用，观察者使用默认位置，即元素中心点对应的平面。

24.1.14　变形原点

2D 变形原点由于没有 Z 轴，所以 Z 轴的值默认为 0。在 3D 变形原点中，Z 轴是一个可以设置的变量。语法格式如下：

```
transform-origin: x 轴  y 轴  z 轴
```

取值简单说明如下：

☑　x 轴：left | center | right | <length> | <percentage>。

☑　y 轴：top | center | bottom | <length> | <percentage>。

☑　z 轴：<length>。

对于 X 轴和 Y 轴来说，可以设置关键字和百分比，分别相对于其本身元素水平方向的宽度和垂直方向的高度；Z 只能设置长度值。



视频讲解

24.1.15　背景可见

元素的背面在默认情况下是可见的，有时可能需要让元素背面不可见，这时候就可以使用 backface-visibility 属性，该属性的具体语法格式如下：

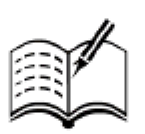


```
backface-visibility:visible | hidden
```

取值简单说明如下：

☑ visible：指定元素背面可见，允许显示正面的镜像，为默认值。

☑ hidden：指定元素背面不可见。



Note

【示例】在 24.1.13 节示例中，如果在变形图像样式中添加“backface-visibility: hidden;”，定义元素背面面向用户时不可见，这时如果再次预览，则会发现变形图像已经不存在，因为它的背面面向用户，被隐藏了，效果如图 24.17 所示。

```
img.change {
    border: solid 1px red;
    transform:perspective(180px) rotateX(120deg);
    backface-visibility: hidden;
}
```

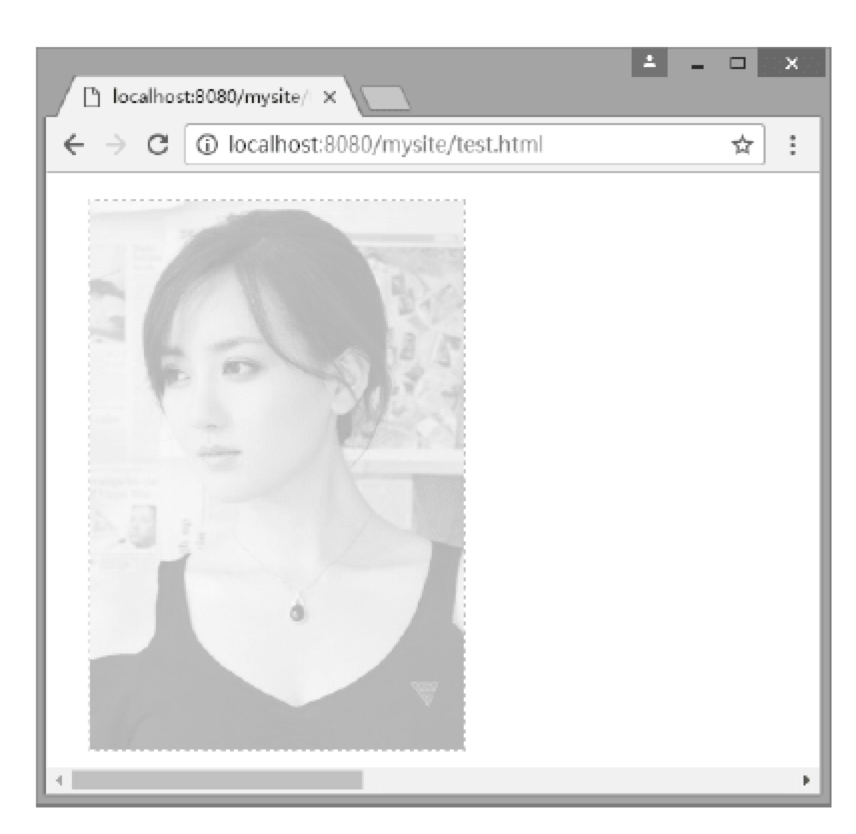


图 24.17　定义背面面向用户不可见效果

24.2　过 渡 动 画

2013 年 2 月，W3C 发布了 CSS Transitions 工作草案，在这个草案中描述了 CSS 过渡动画的基本实现方法和属性。目前已获得所有浏览器的支持，包括支持带有前缀（私有属性）或不带前缀的过渡（标准属性）。最新版本浏览器（IE 10+、Firefox 16+和 Opera 12.5+）均支持不带前缀的过渡属性 transition，而旧版浏览器则支持前缀的过渡，如 Webkit 引擎支持-webkit-transition 私有属性，Mozilla Gecko 引擎支持-moz-transition 私有属性，Presto 引擎支持-o-transition 私有属性，IE 6～IE 9 浏览器不支持 transition 属性，IE 10 支持 transition 属性。

权威参考：http://www.w3.org/TR/css3-transitions/。

权 威 参 考



视 频 讲 解

24.2.1　设置过渡属性

transition-property 属性用来定义过渡动画的 CSS 属性名称，基本语法如下所示：

```
transition-property:none | all | [ <IDENT> ] [ ','<IDENT> ]*;
```



取值简单说明如下：

☑ none：表示没有元素。

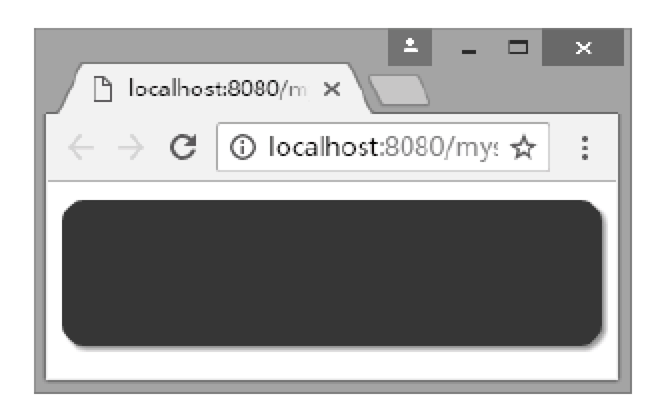
☑ all：默认值，表示针对所有元素，包括“:before”和“:after”伪元素。

☑ IDENT：指定 CSS 属性列表。几乎所有色彩、大小或位置等相关的 CSS 属性，包括许多新添加的 CSS3 属性，都可以应用过渡，如 CSS3 变换中的放大、缩小、旋转、斜切、渐变等。

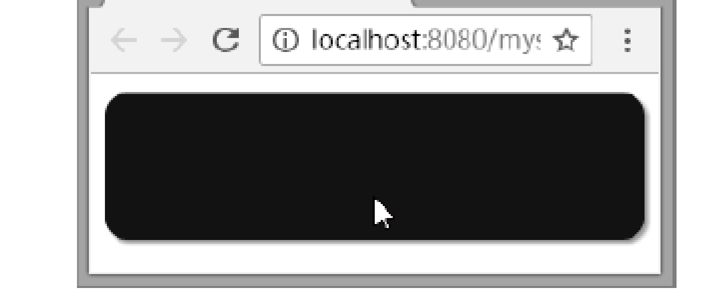
【示例】在下面示例中，指定动画的属性为背景色。这样当鼠标经过盒子时，会自动从红色背景过渡到蓝色背景，演示效果如图 24.18 所示。

```
<style type="text/css">
div {
    margin: 10px auto; height: 80px;
    background: red;
    border-radius: 12px;
    box-shadow: 2px 2px 2px #999;
}
div:hover {
    background-color: blue;
    /*指定动画过渡的 CSS 属性*/
    transition-property: background-color;
}
</style>

<div></div>
```



（a）默认状态



（b）鼠标经过时被旋转

图 24.18　定义简单的背景色切换动画



视频讲解

24.2.2　设置过渡时间

transition-duration 属性用来定义转换动画的时间长度，基本语法如下所示：

```
transition-duration:<time> [, <time>]*;
```

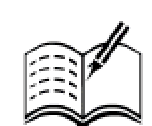
初始值为 0，适用于所有元素，以及“:before”和“:after”伪元素。在默认情况下，动画过渡时间为 0 秒，所以当指定元素动画时，会看不到过渡的过程，直接看到结果。

【示例】以 24.2.1 节示例为例，下面示例设置动画过渡时间为 2 秒，当鼠标移过对象时，会看到背景色从红色逐渐过渡到蓝色，演示效果如图 24.19 所示。

```
div:hover {
    background-color: blue;
    /*指定动画过渡的 CSS 属性*/
    transition-property: background-color;
```



```
        /*指定动画过渡的时间*/
        transition-duration:2s;
    }
```



Note

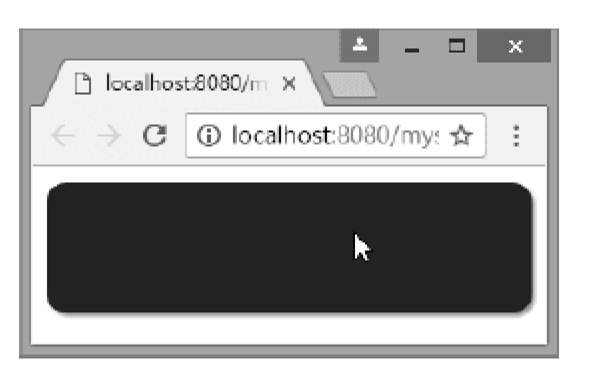


图 24.19　设置动画时间

24.2.3　设置延迟过渡时间



视频讲解

transition-delay 属性用来定义开启过渡动画的延迟时间，基本语法如下所示：

```
transition-delay:<time> [, <time>]*;
```

初始值为 0，适用于所有元素，以及“:before”和“:after”伪元素。设置时间可以为正整数、负整数和零，非零的时候必须设置单位是 s（秒）或者 ms（毫秒），为负数的时候，过渡的动作会从该时间点开始显示，之前的动作被截断。为正数的时候，过渡的动作会延迟触发。

【示例】继续以 24.2.1 节示例为基础进行介绍，下面示例设置过渡动画推迟 2 秒钟后执行，则当鼠标移过对象时，会看不到任何变化，过了 2 秒钟之后，才发现背景色从红色逐渐过渡到蓝色。

```
div:hover {
    background-color: blue;
    /*指定动画过渡的 CSS 属性*/
    transition-property: background-color;
    /*指定动画过渡的时间*/
    transition-duration: 2s;
    /*指定动画延迟触发  */
    transition-delay: 2s;
}
```

24.2.4　设置过渡动画类型



视频讲解

transition-timing-function 属性用来定义过渡动画的类型，基本语法如下所示：

```
transition-timing-function:ease | linear | ease-in | ease-out | ease-in-out | cubicbezier(<number>, <number>,
<number>, <number>) [, ease | linear | ease-in | ease-out | ease-in-out | cubic-bezier(<number>, <number>,<number>,
<number>)]*
```

属性初始值为 ease，取值简单说明如下：

☑　ease：平滑过渡，等同于 cubic-bezier(0.25, 0.1, 0.25, 1.0)函数，即立方贝塞尔。

☑　linear：线性过渡，等同于 cubic-bezier(0.0, 0.0, 1.0, 1.0)函数。

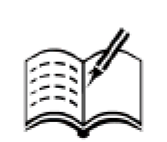
☑　ease-in：由慢到快，等同于 cubic-bezier(0.42, 0, 1.0, 1.0)函数。

☑　ease-out：由快到慢，等同于 cubic-bezier(0, 0, 0.58, 1.0)函数。

☑　ease-in-out：由慢到快再到慢，等同于 cubic-bezier(0.42, 0, 0.58, 1.0)函数。

☑　cubic-bezier：特殊的立方贝塞尔曲线效果。





Note

【示例】继续以 24.2.1 节示例为基础进行介绍，下面设置过渡类型为线性效果，代码如下所示：

```
div:hover {
    background-color: blue;
    /*指定动画过渡的 CSS 属性*/
    transition-property: background-color;
    /*指定动画过渡的时间*/
    transition-duration: 10s;
    /*指定动画过渡为线性效果 */
    transition-timing-function: linear;
}
```



视频讲解

24.2.5 设置过渡触发动作

CSS3 过渡动画一般通过动态伪类触发，如表 24.1 所示。

表 24.1 CSS 动态伪类

动 态 伪 类	作 用 元 素	说 明
:link	只有链接	未访问的链接
:visited	只有链接	访问过的链接
:hover	所有元素	鼠标经过元素
:active	所有元素	鼠标单击元素
:focus	所有可被选中的元素	元素被选中

也可以通过 JavaScript 事件触发，包括 click、focus、mousemove、mouseover、mouseout 等。

1. :hover

最常用的过渡触发方式是使用“:hover”伪类。

【示例 1】下面示例设计当鼠标经过 div 元素时，该元素的背景色会在经过一秒钟的初始延迟后，于两秒钟内动态地从绿色变为蓝色。

```
<style type="text/css">
div {
    margin: 10px auto;
    height: 80px;
    border-radius: 12px;
    box-shadow: 2px 2px 2px #999;
    background-color: red;
    transition: background-color 2s ease-in 1s;
}
div:hover { background-color: blue}
</style>
<div></div>
```

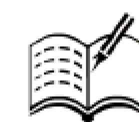
2. :active

“:active”伪类表示用户单击某个元素并按住鼠标按钮时显示的状态。

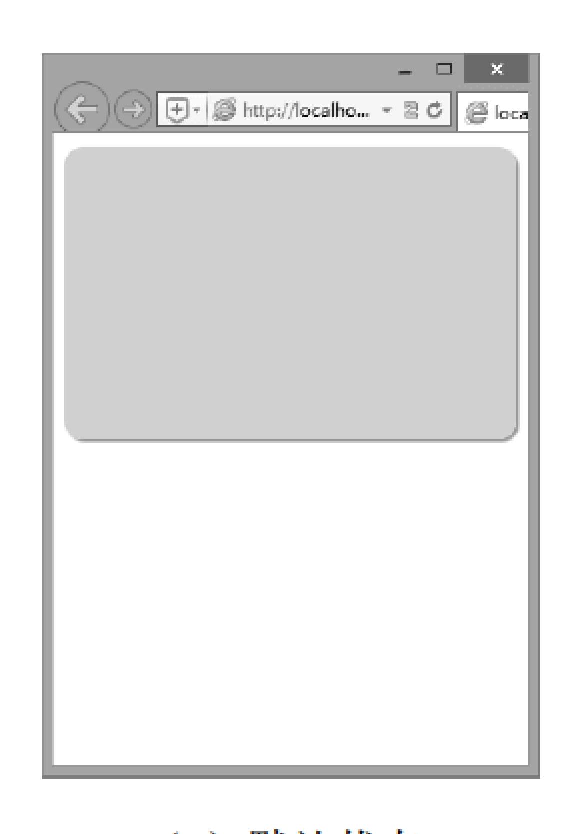
【示例 2】下面示例设计当用户单击 div 元素时，该元素被激活，这时会触发动画，高度属性从



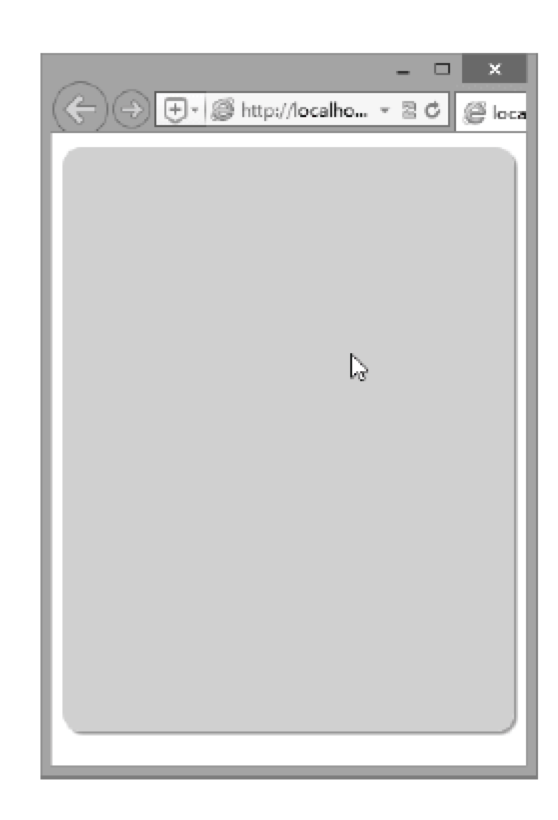
200px 过渡到 400px。如果按住该元素，保持住活动状态，则 div 元素始终显示 400px 高度，松开鼠标之后，又会恢复原来的高度，如图 24.20 所示。



```
<style type="text/css">
div {
    margin: 10px auto;
    border-radius: 12px;
    box-shadow: 2px 2px 2px #999;
    background-color: #8AF435;
    height: 200px;
    transition: width 2s ease-in;
}
div:active {height: 400px;}
</style>
<div></div>
```



（a）默认状态



（b）单击

图 24.20　定义激活触发动画

3. : focus

“:focus”伪类通常会在表单对象接收键盘响应时出现。

【示例 3】下面设计当输入框获取焦点时，输入框的背景色逐步高亮显示，如图 24.21 所示。

```
<style type="text/css">
label {
    display: block;
    margin: 6px 2px;
}
input[type="text"], input[type="password"] {
    padding: 4px;
    border: solid 1px #ddd;
    transition: background-color 1s ease-in;
}
input:focus { background-color: #9FFC54;}
</style>
<form id=fm-form action="" method=post>
    <fieldset>
```



Note

```
            <legend>用户登录</legend>
            <label for="name">姓名
                <input type="text" id="name" name="name" >
            </label>
            <label for="pass">密码
                <input type="password" id="pass" name="pass" >
            </label>
        </fieldset>
</form>
```

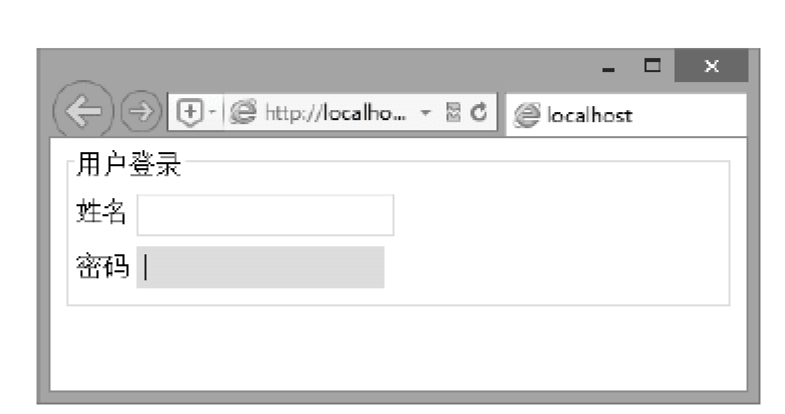


图 24.21　定义获取焦点触发动画

提示：将“:hover”伪类与“:focus”配合使用，能够丰富鼠标用户和键盘用户的体验。

4. :checked

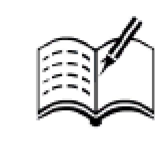
“:checked”伪类在发生这种状况时触发过渡。

【示例 4】下面示例设计当复选框被选中时缓慢缩进两个字符，演示效果如图 24.22 所示。

```
<style type="text/css">
label.name {
    display: block;
    margin: 6px 2px;
}
input[type="text"], input[type="password"] {
    padding: 4px;
    border: solid 1px #ddd;
}
input[type="checkbox"] { transition: margin 1s ease;}
input[type="checkbox"]:checked { margin-left: 2em;}
</style>
<form id=fm-form action="" method=post>
    <fieldset>
        <legend>用户登录</legend>
        <label class="name" for="name">姓名
            <input type="text" id="name" name="name" >
        </label>
        <p>技术专长<br>
            <label>
                <input type="checkbox" name="web" value="html" id="web_0">
                HTML</label><br>
            <label>
                <input type="checkbox" name="web" value="css" id="web_1">
                CSS</label><br>
            <label>
```



```
            <input type="checkbox" name="web" value="javascript" id="web 2">
            JavaScript</label><br>
        </p>
    </fieldset>
</form>
```



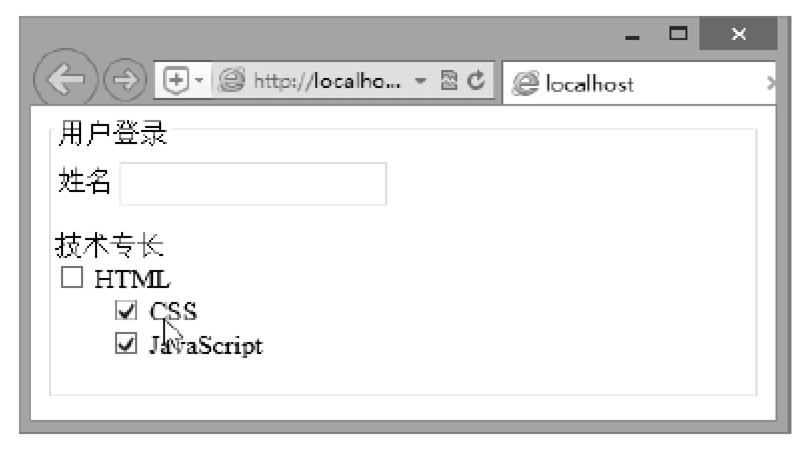


图 24.22　定义被选中时触发动画

5. 媒体查询

触发元素状态变化的另一种方法是使用 CSS3 媒体查询，关于媒体查询详解参考下章内容。

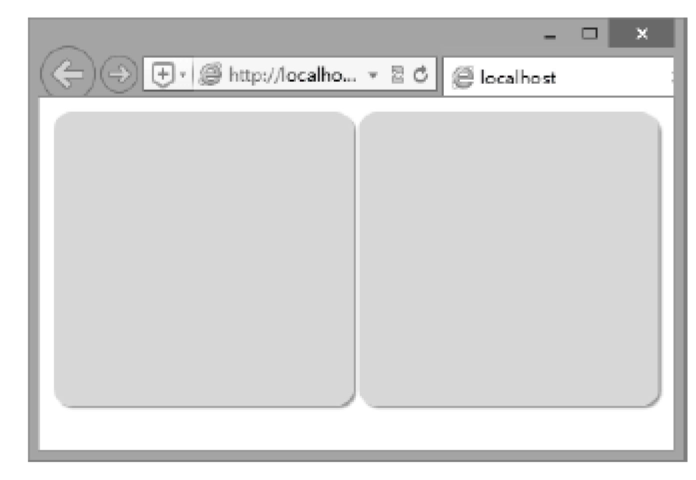
【示例 5】下面示例设计 div 元素的宽度和高度为 49%×200px，如果用户将窗口大小调整到 420px 或以下，则该元素将过渡为 100%×100px。也就是说，当窗口宽度变化经过 420px 的阈值时，将会触发过渡动画，如图 24.23 所示。

```
<style type="text/css">
div {
    float: left; margin: 2px;
    width: 49%; height: 200px;
    background: #93FB40;
    border-radius: 12px;
    box-shadow: 2px 2px 2px #999;
    transition: width 1s ease, height 1s ease;
}
@media only screen and (max-width : 420px) {
    div {
        width: 100%;
        height: 100px;
    }
}
</style>
<div></div>
<div></div>
```





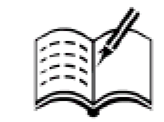
（a）当窗口小于等于 420px 宽度



（b）当窗口大于 420px 宽度

图 24.23　设备类型触发动画





Note

如果网页加载时用户的窗口大小是 420px 或以下，浏览器会在该部分应用这些样式，但是由于不会出现状态变化，因此不会发生过渡。

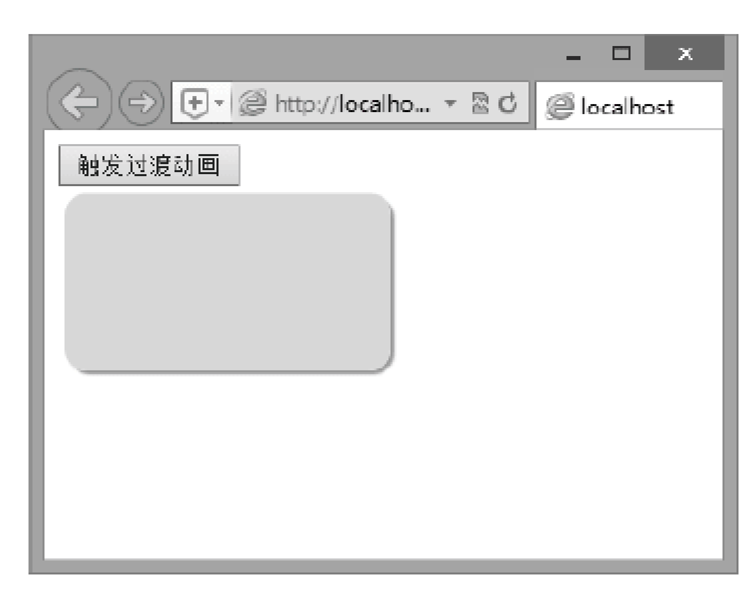
6. JavaScript 事件

【示例 6】下面示例可以使用纯粹的 CSS 伪类触发过渡，为了方便用户理解，这里通过 jQuery 脚本触发过渡。

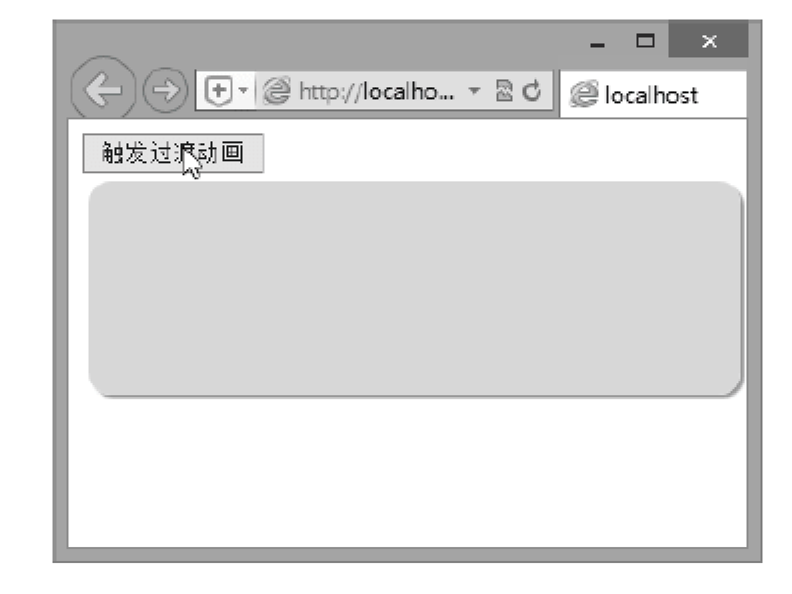
```
<script type="text/javascript" src="images/jquery-1.10.2.js"></script>
<script type="text/javascript">
$(function() {
    $("#button").click(function() {
        $(".box").toggleClass("change");
    });
});
</script>
<style type="text/css">
.box {
    margin:4px;
    background: #93FB40;
    border-radius: 12px;
    box-shadow: 2px 2px 2px #999;
    width: 50%; height: 100px;
    transition: width 2s ease, height 2s ease;
}
.change { width: 100%; height: 120px;}
</style>
<input type="button" id="button" value="触发过渡动画" />
<div class="box"></div>
```

在文档中包含一个 box 类的盒子和一个按钮，当单击按钮时，jQuery 脚本会将盒子的类切换为 change，从而触发过渡动画，演示效果如图 24.24 所示。

示 例 效 果



（a）默认状态



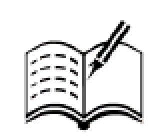
（b）JavaScript 事件激活状态

图 24.24　使用 JavaScript 脚本触发动画

上面演示了样式发生变化会导致过渡动画，也可以通过其他方法触发这些更改，包括通过 JavaScript 脚本动态更改。从执行效率来看，事件通常应当通过 JavaScript 触发，简单动画或过渡则应使用 CSS 触发。



24.3　帧　动　画



Note

2012 年 4 月，W3C 发布了 CSS Animations 工作草案，在这个草案中描述了 CSS 关键帧动画的基本实现方法和属性。目前最新版本的主流浏览器都支持 CSS 帧动画，如 IE 10+、Firefox 和 Opera 浏览器均支持不带前缀的动画属性 animation，而旧版浏览器则支持前缀的动画，如 Webkit 引擎支持-webkit-animation 属性，Mozilla Gecko 引擎支持-moz-animation 私有属性，Presto 引擎支持-o-animation 私有属性，IE 6～IE 9 浏览器不支持 animation 属性。

权威参考：http://www.w3.org/TR/css3-animations/。

权威参考



视频讲解

24.3.1　设置关键帧

CSS3 使用@keyframes 定义关键帧。具体用法如下所示：

```
@keyframes animationname {
    keyframes-selector {
        css-styles;
    }
}
```

其中参数说明如下：

☑ animationname：定义动画的名称。

☑ keyframes-selector：定义帧的时间未知，也就是动画时长的百分比，合法的值包括：0～100%、from（等价于 0%）、to（等价于 100%）。

☑ css-styles：表示一个或多个合法的 CSS 样式属性。

在动画过程中，用户能够多次改变这套 CSS 样式。以百分比来定义样式改变发生的时间，或者通过关键词 from 和 to。为了获得最佳浏览器支持，设计关键帧动画时，应该始终定义 0 和 100%位置帧。最后，为每帧定义动态样式，同时将动画与选择器绑定。

【示例】下面示例演示如何让一个小方盒沿着方形框内壁匀速运动，效果如图 24.25 所示。

```
<style>
#wrap {                         /* 定义运动轨迹包含框*/
    position:relative;          /* 定义定位包含框，避免小盒子跑到外面运动*/
    border:solid 1px red;
    width:250px; height:250px;
}
#box {                          /* 定义运动小盒的样式*/
    position:absolute;
    left:0; top:0;
    width: 50px; height: 50px;
    background: #93FB40;
    border-radius: 8px;
    box-shadow: 2px 2px 2px #999;
    /*定义帧动画：名称为 ball，动画时长 5 秒，动画类型为匀速渐变，动画无限播放*/
    animation: ball 5s linear infinite;
}
```



```
/*定义关键帧：共包括 5 帧，分别在总时长 0、25%、50%、75%、100%的位置*/
/*每帧中设置动画属性为 left 和 top，让它们的值匀速渐变，产生运动动画*/
@keyframes ball {
    0% {left:0;top:0;}
    25% {left:200px;top:0;}
    50% {left:200px;top:200px;}
    75% {left:0;top:200px;}
    100% {left:0;top:0;}
}
</style>
<div id="wrap">
    <div id="box"></div>
</div>
```



示例效果

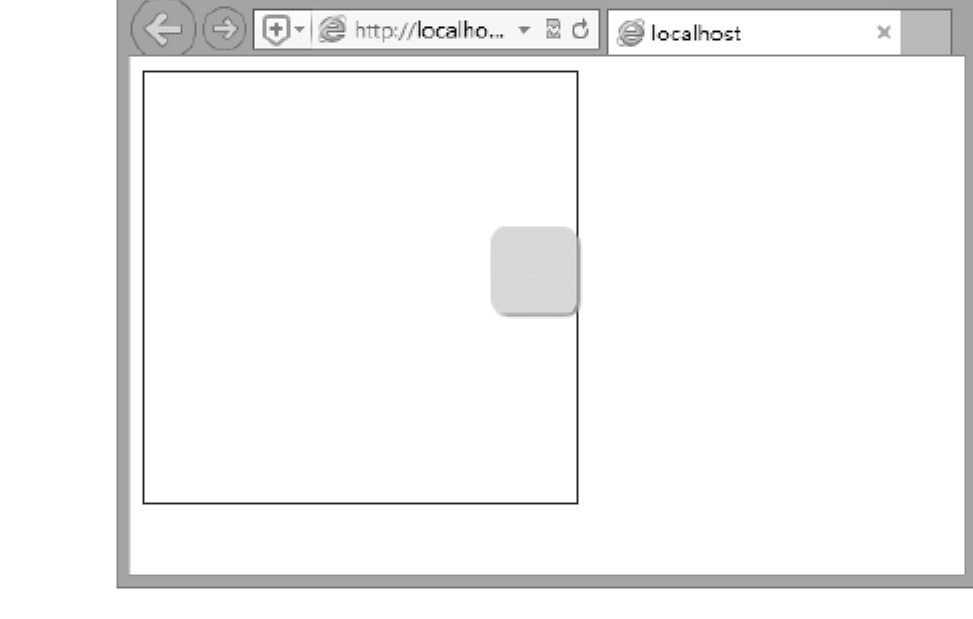


图 24.25　设计小盒子运动动画



视频讲解

24.3.2　设置动画属性

Animations 功能与 Transitions 功能相同，都是通过改变元素的属性值来实现动画效果。它们的区别在于：使用 Transitions 功能时只能通过指定属性的开始值与结束值，然后在这两个属性值之间进行平滑过渡的方式来实现动画效果，因此不能实现比较复杂的动画效果；而 Animations 则通过定义多个关键帧以及定义每个关键帧中元素的属性值来实现更为复杂的动画效果。

1. 定义动画名称

使用 animation-name 属性可以定义 CSS 动画的名称，语法如下所示：

```
animation-name:none | IDENT [, none | IDENT ]*;
```

初始值为 none，定义一个适用的动画列表。每个名字用来选择动画关键帧，提供动画的属性值。如名称是 none，那么就不会有动画。

2. 定义动画时间

使用 animation-duration 属性可以定义 CSS 动画播放时间，语法如下所示：

```
animation-duration:<time> [, <time>]*;
```

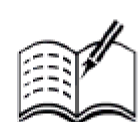
在默认情况下该属性值为 0，这意味着动画周期是直接的，即不会有动画。当值为负值时，则被视为 0。



3．定义动画类型

使用 animation-timing-function 属性可以定义 CSS 动画类型，语法如下所示：

```
animation-timing-function:ease | linear | ease-in | ease-out | ease-in-out | cubicbezier(<number>, <number>, number>, <number>) [, ease | linear |ease-in | ease-out | ease-in-out | cubic-bezier(<number>, <number>,<number>, <number>)]*
```



初始值为 ease，取值说明可参考上面介绍的过渡动画类型。

4．定义延迟时间

使用 animation-delay 属性可以定义 CSS 动画延迟播放的时间，语法如下所示：

```
animation-delay:<time> [, <time>]*;
```

该属性允许一个动画开始执行一段时间后才被应用。当动画延迟时间为 0，即默认动画延迟时间，则意味着动画将尽快执行，否则该值指定将延迟执行的时间。

5．定义播放次数

使用 animation-iteration-count 属性定义 CSS 动画的播放次数，语法如下所示：

```
animation-iteration-count:infinite | <number> [, infinite | <number>]*;
```

默认值为 1，这意味着动画将从开始到结束播放一次。infinite 表示无限次，即 CSS 动画永远重复。如果取值为非整数，将导致动画结束一个周期的一部分。如果取值为负值，则将导致在交替周期内反向播放动画。

6．定义播放方向

使用 animation-direction 属性定义 CSS 动画的播放方向，基本语法如下所示：

```
animation-direction:normal | alternate [, normal | alternate]*;
```

默认值为 normal。当为默认值时，动画的每次循环都向前播放。另一个值是 alternate，设置该值则表示第偶数次向前播放，第奇数次向反方向播放。

7．定义播放状态

使用 animation-play-state 属性定义动画正在运行还是暂停，语法如下所示：

```
animation-play-state: paused|running;
```

初始值为 running。其中 paused 定义动画已暂停，running 定义动画正在播放。

提示：可以在 JavaScript 中使用该属性，这样就能在播放过程中暂停动画。在 JavaScript 脚本中用法如下：

```
object.style.animationPlayState="paused"
```

8．定义播放外状态

使用 animation-fill-mode 属性定义动画外状态，语法如下所示：

```
animation-fill-mode:none | forwards | backwards | both [ , none | forwards | backwards | both ]*
```

初始值为 none，如果提供多个属性值，以逗号进行分隔。取值说明如下：



☑ none：不设置对象动画之外的状态。

☑ forwards：设置对象状态为动画结束时的状态。

☑ backwards：设置对象状态为动画开始时的状态。

☑ both：设置对象状态为动画结束或开始时的状态。

【示例】下面示例设计一个小球，并定义它水平向左运动，动画结束之后，再返回起始点位置，效果如图 24.26 所示。

```
<style>
/*启动运动的小球，并定义动画结束后返回*/
.ball{
    width: 50px; height: 50px;
    background: #93FB40;
    border-radius: 100%;
    box-shadow:2px 2px 2px #999;
    animation:ball 1s ease backwards;
}
/*定义小球水平运动关键帧*/
@keyframes ball{
    0%{transform:translate(0,0);}
    100%{transform:translate(400px);}
}
</style>
<div class="ball"></div>
```



示例效果

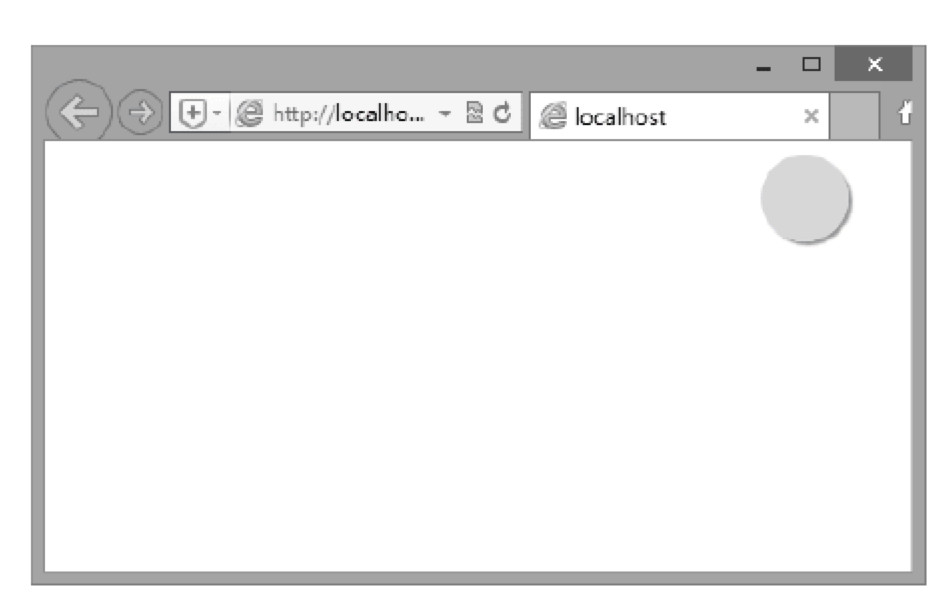


图 24.26　设计运动小球最后返回起始点位置

24.4　案例实战

本节将通过多个案例帮助读者上机练习和提升 CSS3 动画设计技法。

24.4.1　设计图形



线上阅读



视频讲解

设计菱形，效果如图 24.27 所示；设计平行四边形，效果如图 24.28 所示。

设计星形，效果如图 24.29 所示；设计心形，效果如图 24.30 所示。具体代码解析请扫码学习。



Note

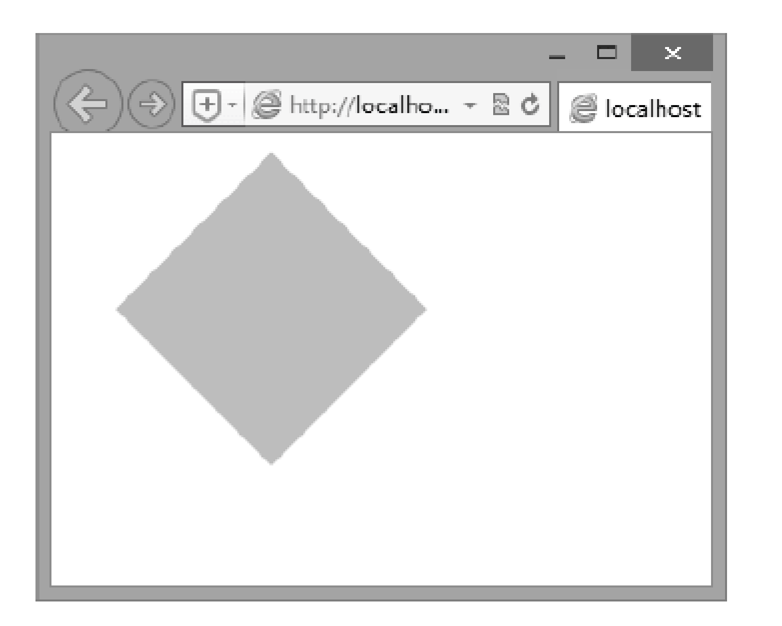


图 24.27　设计菱形

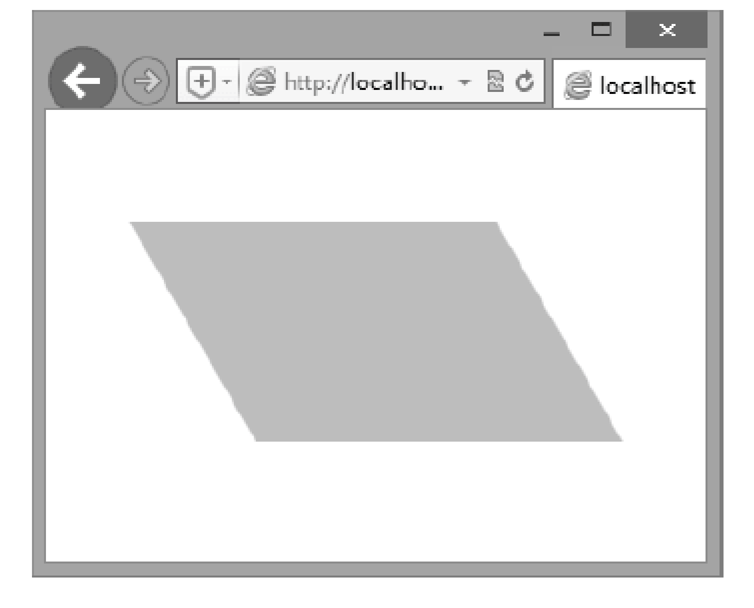


图 24.28　设计平行四边形

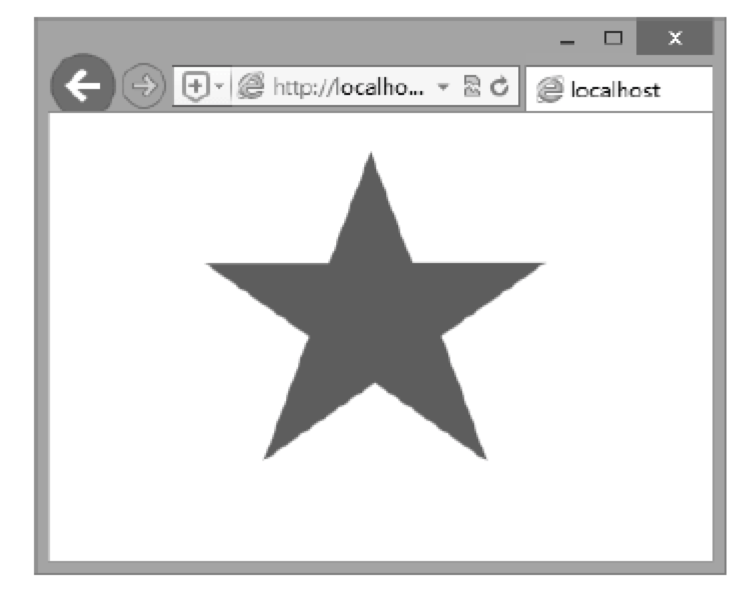


图 24.29　设计星形

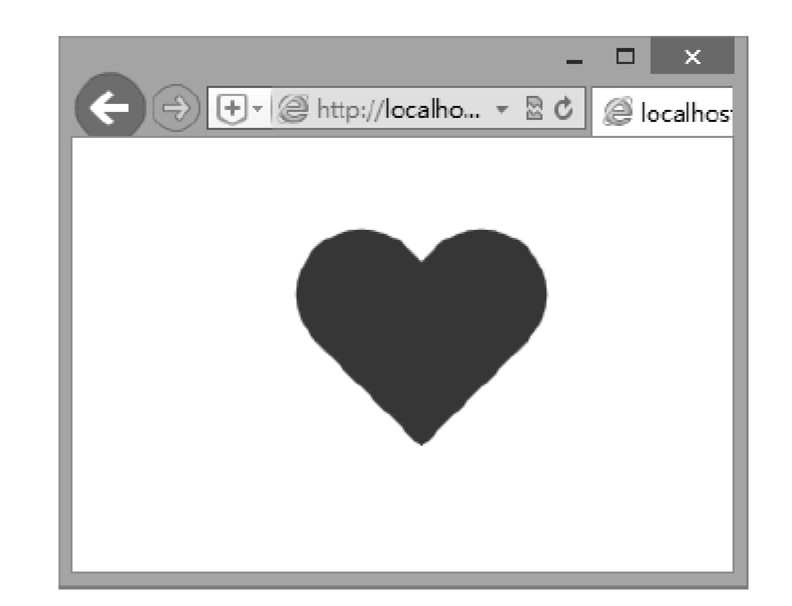


图 24.30　设计心形

24.4.2　设计冒泡背景按钮

本例应用 CSS3 过渡动画特效，为按钮背景图像定义动态移动效果，设计当鼠标经过时，按钮背景绚丽多彩，不断产生冒泡动画效果，如图 24.31 所示。

具体操作步骤请扫码学习。





线上阅读　视频讲解

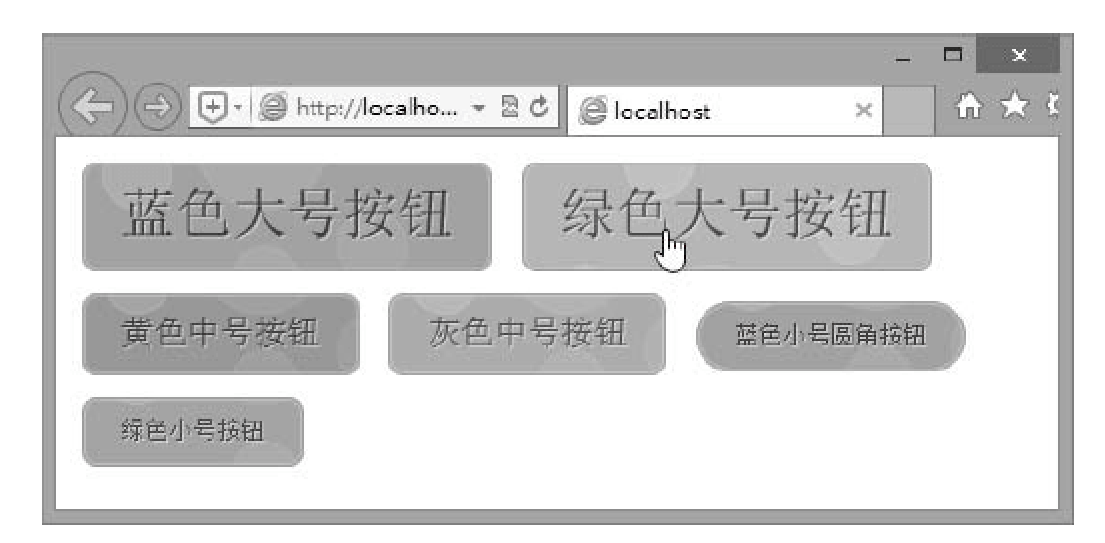


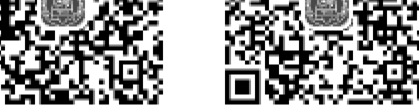
图 24.31　设计背景冒泡效果的按钮样式

24.4.3　设计动画效果菜单

本例利用 CSS3 过渡动画设计一个界面切换的导航菜单，当鼠标经过菜单项时，会以动画形式从中文界面缓慢翻转到英文界面，或者从英文界面翻转到中文界面，效果如图 24.32 所示。

具体操作步骤请扫码学习。





线上阅读　视频讲解






图 24.32　设计动画翻转菜单样式

24.4.4　设计照片特效


线上阅读

视频讲解

本例使用 CSS3 阴影、透明效果以及变换，让图片随意贴在墙上，当鼠标移动到图片上时，会自动放大并垂直摆放，演示效果如图 24.33 所示。在默认状态下，图片被随意显示在墙面上，鼠标经过图片时，图片会竖直摆放，并被放大显示。

具体代码解析请扫码学习。

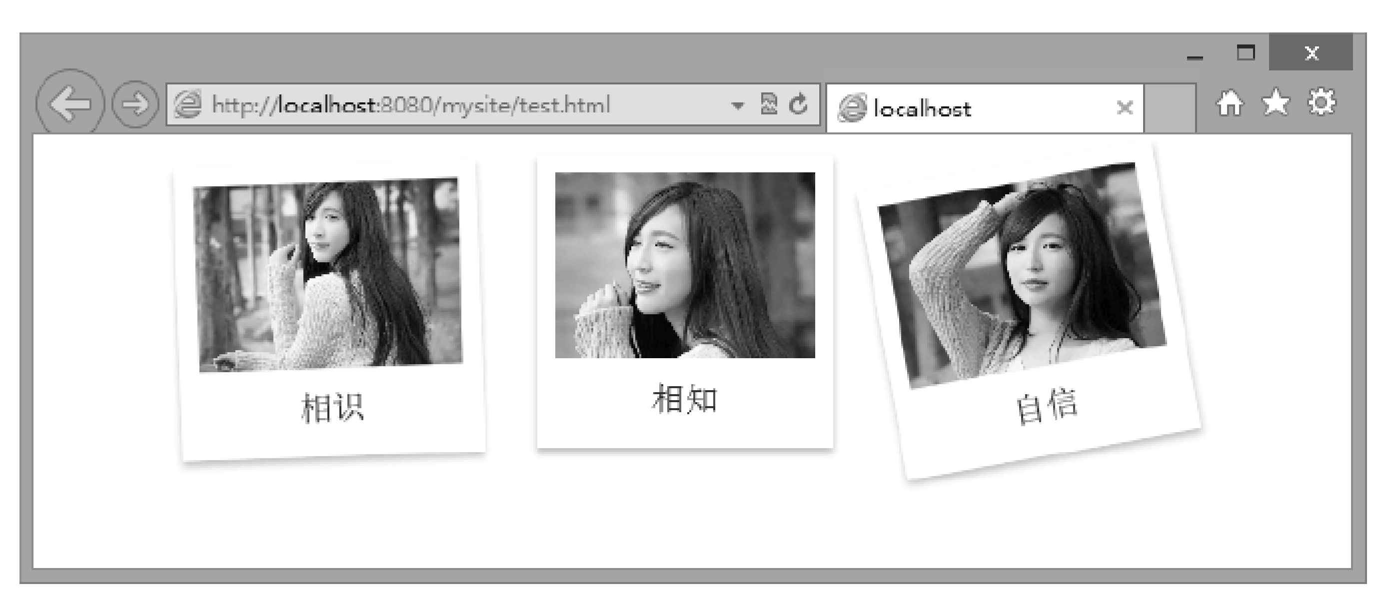


图 24.33　设计挂图效果

24.4.5　设计立体盒子


线上阅读

视频讲解

使用 2D 多重变换制作一个正方体，演示效果如图 24.34 所示；使用 3D 多重变换制作一个正方体，演示效果如图 24.35 所示。

具体代码解析请扫码学习。



图 24.34　设计 2D 变换盒子

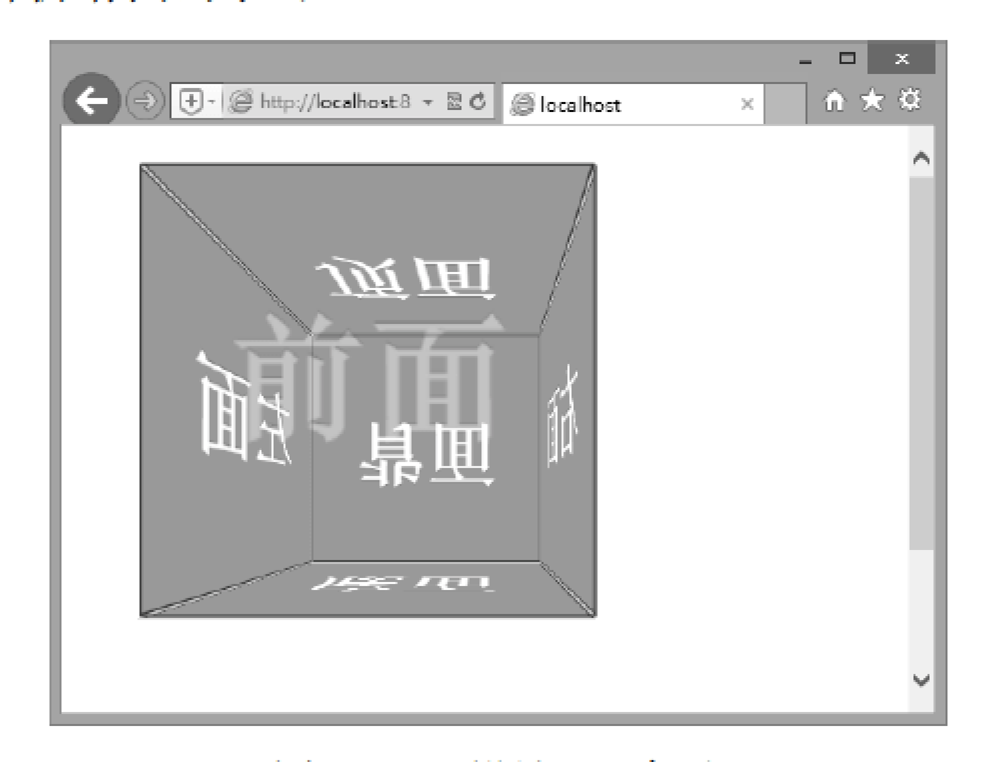


图 24.35　设计 3D 盒子



Note

24.4.6　旋转盒子

继续以 24.4.5 节示例为基础，本节示例设计使用 animation 属性设计盒子旋转显示。

具体说明和操作步骤请扫码学习。




线上阅读　视频讲解



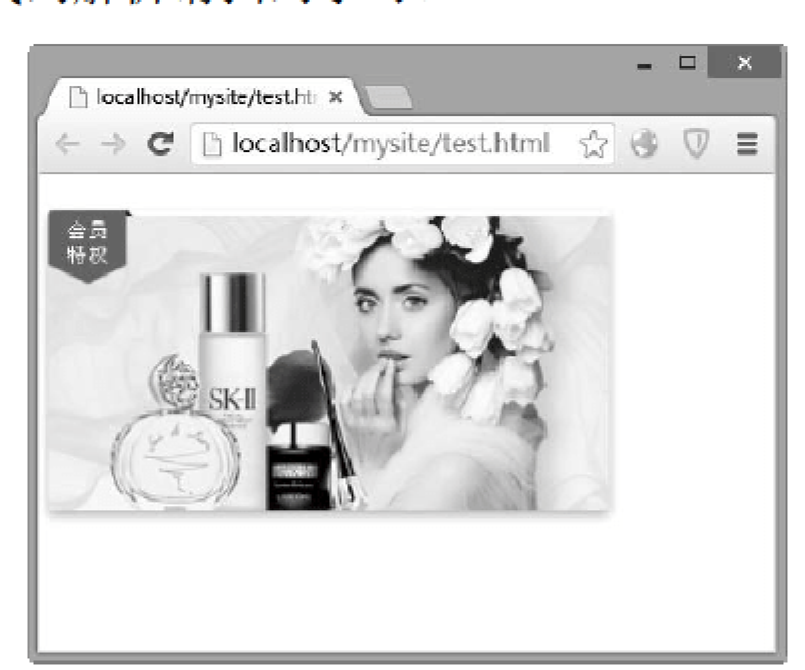
24.4.7　设计翻转广告

本例设计当鼠标移动到产品图片上时，产品信息翻转滑出，效果如图 24.36 所示。在默认状态下只显示产品图片，而产品信息隐藏不可见。当用户鼠标移动到产品图像上时，产品图像慢慢往上旋转使产品信息展示出来，而产品图像慢慢隐藏起来，看起来就像是一个旋转的盒子。

具体代码解析请扫码学习。




线上阅读　视频讲解



（a）默认状态



（b）翻转状态

图 24.36　设计 3D 翻转广告牌

24.4.8　设计跑步效果

本例设计一个跑步动画效果，主要使用 CSS3 帧动画控制一张序列人物跑步的背景图像，在页面固定“镜头”中快速切换实现动画效果，如图 24.37 所示。

具体操作步骤请扫码学习。



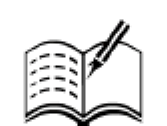

线上阅读　视频讲解



图 24.37　设计跑步的小人



24.4.9 设计折叠面板



Note



线上阅读



视频讲解

本例使用 CSS3 的目标伪类（:target）设计折叠面板效果，没有使用 JavaScript 脚本，使用过渡属性设计滑动效果，折叠动画效果如图 24.38 所示。

具体代码解析请扫码学习。



图 24.38 设计折叠面板

24.5 在线练习

练习 CSS3 动画一般设计方法，培养灵活应用交互式动态样式的基本能力。



在线练习

第 25 章

CSS3 媒体查询

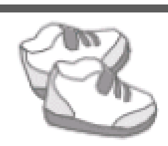
2017 年 9 月，W3C 发布了媒体查询（Media Query Level 4）候选推荐标准规范，它扩展了已经发布的媒体查询的功能。该规范用于 CSS 的@media 规则，可以为文档设定特定条件的样式，也可用于 HTML、JavaScript 等语言。

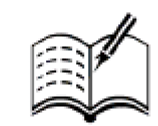
权威参考：http://www.w3.org/TR/css3-mediaqueries/

权威参考

【学习重点】

- 了解 CSS3 媒体类型。
- 正常使用媒体查询的条件规则。
- 设计响应不同设备的网页布局。





Note

25.1　媒体查询基础

媒体查询可以根据设备特性，如屏幕宽度、高度、设备方向（横向或纵向），为设备定义独立的CSS样式表。一个媒体查询由一个可选的媒体类型和零个或多个限制范围的表达式组成，如宽度、高度和颜色。

25.1.1　媒体类型和媒体查询

CSS 2提出媒体类型（Media Type）的概念，它允许为样式表设置限制范围的媒体类型。例如，仅供打印的样式表文件、仅供手机渲染的样式表文件、仅供电视渲染的样式表文件等，具体说明如表25.1所示。

表25.1　CSS媒体类型

类　型	支持的浏览器	说　明
aural	Opera	用于语音和音乐合成器
braille	Opera	用于触觉反馈设备
handheld	Chrome,Safari,Opera	用于小型或手持设备
print	所有浏览器	用于打印机
projection	Opera	用于投影图像，如幻灯片
screen	所有浏览器	用于屏幕显示器
tty	Opera	用于使用固定间距字符格的设备。如电传打字机和终端
tv	Opera	用于电视类设备
embossed	Opera	用于凸点字符（盲文）印刷设备
speech	Opera	用于语音类型
all	所有浏览器	用于所有媒体设备类型

通过HTML标签属性media属性定义样式表的媒体类型，具体方法如下：

☑　定义外部样式表文件的媒体类型。

```
<link href="csss.css" rel="stylesheet" type="text/css" media="handheld" />
```

☑　定义内部样式表文件的媒体类型。

```
<style type="text/css" media="screen">
...
</style>
```

CSS3在媒体类型基础上，提出了Media Queries（媒体查询）的概念。媒体查询比CSS2的媒体类型功能更强大、更加完善。两者主要区别：媒体查询是一个值或一个范围的值，而媒体类型仅仅是设备的匹配。媒体类型可以帮助用户获取以下数据。

☑　浏览器窗口的宽和高。

☑　设备的宽和高。

☑　设备的手持方向，横向还是竖向。



☑　分辨率。

例如，下面这条导入外部样式表的语句。

```
<link rel="stylesheet" media="screen and (max-width: 600px)" href="small.css" />
```

在 media 属性中设置媒体查询的条件“(max-width: 600px)”：当屏幕宽度小于或等于 600px，则调用 small.css 样式表来渲染页面。

25.1.2　使用@media

CSS3 使用@media 规则定义媒体查询，简化语法格式如下：

```
@media [only | not]? <media_type> [and <expression>]* | <expression> [and <expression>]*{
    /* CSS 样式列表 */
}
```

参数简单说明如下：

☑　<media_type>：指定媒体类型，具体说明参考表 25.1。

☑　<expression>：指定媒体特性。放在一对圆括号中，如“(min-width:400px)”。

☑　逻辑运算符，如 and（逻辑与）、not（逻辑否）、only（兼容设备）等。

媒体特性包括 13 种，接受单个的逻辑表达式作为值，或者没有值。大部分特性接受 min 或 max 的前缀，用来表示大于等于，或者小于等于的逻辑，以此避免使用大于号（>）和小于号（<）字符。



线 上 阅 读

各种媒体特性的简单说明请扫码了解。

在 CSS 样式的开头必须定义@media 关键字，然后指定媒体类型，再指定媒体特性。媒体特性的格式与样式的格式相似，分为两部分，由冒号分隔，冒号前指定媒体特性，冒号后指定该特性的值。

例如，下面语句指定了当设备显示屏幕宽度小于 640px 时所使用的样式。

```
@media screen and (max-width: 639px) {
    /*样式代码*/
}
```

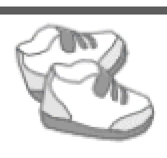
可以使用多个媒体查询将同一个样式应用于不同的媒体类型和媒体特性中，媒体查询之间通过逗号分隔，类似于选择器分组。

```
@media handheld and (min-width:360px),screen and (min-width:480px) {
    /*样式代码*/
}
```

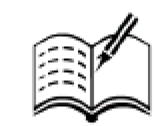
可以在表达式中加上 not、only 和 and 等逻辑运算符。

```
//下面样式代码将被使用在除便携设备之外的其他设备或非彩色便携设备中
@media not handheld and (color) {
    /*样式代码*/
}
//下面样式代码将被使用在所有非彩色设备中
@media all and (not color) {
    /*样式代码*/
}
```

only 运算符能够让那些不支持媒体查询，但是支持媒体类型的设备忽略表达式中的样式。例如：



```
@media only screen and (color) {
    /*样式代码*/
}
```



支持媒体查询的设备，能够正确地读取其中的样式，仿佛 only 运算符不存在一样；对于不支持媒体查询，但支持媒体类型的设备（如 IE 8）来说，可以识别@media screen 关键字，但是由于先读取的是 only 运算符，而不是 screen 关键字，将忽略这个样式。

提示： 媒体查询也可以用在@import 规则和<link>标签中。例如：

```
@import url(example.css) screen and (width:800px);
//下面代码定义了如果页面通过屏幕呈现，且屏幕宽度不超过 480px，则加载 shetland.css 样式表
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 480px)" href="shetland.css" />
```

25.1.3 应用@media

【示例 1】and 运算符用于符号两边规则均满足条件的匹配。

```
@media screen and (max-width : 600px) {
    /*匹配宽度小于等于 600px 的屏幕设备*/
}
```

【示例 2】not 运算符用于取非，所有不满足该规则的均匹配。

```
@media not print {
    /*匹配除了打印机以外的所有设备*/
}
```

注意： not 仅应用于整个媒体查询。

```
@media not all and (max-width : 500px) {}
/*等价于*/
@media not (all and (max-width : 500px)) {}
/*而不是*/
@media (not all) and (max-width : 500px) {}
```

在逗号媒体查询列表中，not 仅会否定它所在的媒体查询，而不影响其他媒体查询。

如果在复杂的条件中使用 not 运算符，要显式添加小括号，避免歧义。

【示例 3】“,”（逗号）相当于 or 运算符，用于两边有一条满足则匹配。

```
@media screen , (min-width : 800px) {
    /*匹配屏幕或者宽度大于等于 800px 的设备*/
}
```

【示例 4】在媒体类型中，all 是默认值，匹配所有设备。

```
@media all {
    /*可以过滤不支持 media 的浏览器*/
}
```

常用的媒体类型还有：screen 匹配屏幕显示器，print 匹配打印输出，更多媒体类型可以参考表 25.1。



【示例 5】使用媒体查询时，必须要加括号，一个括号就是一个查询。

```
@media (max-width : 600px) {
    /*匹配界面宽度小于等于 600px 的设备*/
}
@media (min-width : 400px) {
    /*匹配界面宽度大于等于 400px 的设备*/
}
@media (max-device-width : 800px) {
    /*匹配设备（不是界面）宽度小于等于 800px 的设备*/
}
@media (min-device-width : 600px) {
    /*匹配设备（不是界面）宽度大于等于 600px 的设备*/
}
```

提示： 在设计手机网页时，应该使用 device-width/device-height，因为手机浏览器默认会对页面进行一些缩放，如果按照设备宽和高来进行匹配，会更接近预期的效果。

【示例 6】媒体查询允许相互嵌套，这样可以优化代码，避免冗余。

```
@media not print {
    /*通用样式*/
    @media (max-width:600px) {
        /*此条匹配宽度小于等于 600px 的非打印机设备 */
    }
    @media (min-width:600px) {
        /*此条匹配宽度大于等于 600px 的非打印机设备 */
    }
}
```

【示例 7】在设计响应式页面时，用户应该根据实际需要，先确定自适应分辨率的阈值，也就是页面响应的临界点。

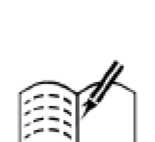
```
@media (min-width: 768px){
    /* >=768px 的设备 */
}
@media (min-width: 992px){
    /* >=992px 的设备 */
}
@media (min-width: 1200){
    /* >=1200px 的设备 */
}
```

注意： 下面样式顺序是错误的，因为后面的查询范围将覆盖掉前面的查询范围，导致前面的媒体查询失效。

```
@media (min-width: 1200){ }
@media (min-width: 992px){ }
@media (min-width: 768px){   }
```

因此，当我们使用 min-width 媒体特性时，应该按从小到大的顺序设计各个阈值。同理，如果使用 max-width，就应该按从大到小的顺序设计各个阈值。





Note

```
@media (max-width: 1199){
    /* <=1199px 的设备 */
}
@media (max-width: 991px){
    /* <=991px 的设备 */
}
@media (max-width: 767px){
    /* <=768px 的设备 */
}
```

【示例 8】用户可以创建多个样式表，以适应不同媒体类型的宽度范围。当然，更有效率的方法是将多个媒体查询整合在一个样式表文件中，这样可以减少请求的数量。

```
@media only screen   and (min-device-width : 320px)   and (max-device-width : 480px) {
    /*样式列表 */
}
@media only screen   and (min-width : 321px) {
    /*样式列表 */
}
@media only screen   and (max-width : 320px) {
    /*样式列表 */
}
```

【示例 9】如果从资源的组织和维护的角度考虑，可以选择使用多个样式表的方式来实现媒体查询，这样做更高效。

```
<link rel="stylesheet" media="screen and (max-width: 600px)" href="small.css" />
<link rel="stylesheet" media="screen and (min-width: 600px)" href="large.css" />
<link rel="stylesheet" media="print" href="print.css" />
```

【示例 10】使用 orientation 属性可以判断设备屏幕当前是横屏（值为 landscape）还是竖屏（值为 portrait）。

```
@media screen and (orientation: landscape) {
    .iPadLandscape {
        width: 30%;
        float: right;
    }
}
@media screen and (orientation: portrait) {
    .iPadPortrait {clear: both;}
}
```

不过 orientation 属性只在 iPad 上有效，对于其他可转屏的设备（如 iPhone），可以使用 min-device-width 和 max-device-width 来变通实现。

【拓展】

媒体查询仅是一种纯 CSS 方式实现响应式 Web 设计的方法，用户还可以使用 JavaScript 库来实现同样的设计。例如，下载 css3-mediaqueries.js（http://code.google.com/p/css3- mediaqueries-js/），然后在页面中调用。对于旧版浏览器（如 IE6、7、8）可以考虑使用 css3-mediaqueries.js 进行兼容。



```
<!–[if lt IE 9]>
<script src=”http://css3-mediaqueries-js.googlecode.com/svn/trunk/css3-mediaqueries.js”></script>
<![endif]–>
```



【示例 11】下面代码演示了如何使用 jQuery 来检测浏览器宽度，并为不同的视口调用不同的样式表。

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $(window).bind("resize", resizeWindow);
    function resizeWindow(e){
        var newWindowWidth = $(window).width();
        if(newWindowWidth < 600){
            $("link[rel=stylesheet]").attr({href : "mobile.css"});
        }
        else if(newWindowWidth > 600){
            $("link[rel=stylesheet]").attr({href : "style.css"});
        }
    }
});
</script>
```

25.2　案 例 实 战

本节将通过几个实例练习 CSS3 媒体查询的网页应用。

25.2.1　判断显示屏幕宽度



视 频 讲 解

本示例设计当显示屏幕宽度小于等于 600px 时，则高亮显示<div class="wrapper b">测试条，并在底部显示提示信息：小于等于 600px；当显示屏幕宽度介于 600px 和 900px 之间时，则高亮显示<div class="wrapper c">测试条，并在底部显示提示信息：介于 600px 和 900px 之间；显示屏幕宽度大于等于 900px 时，则高亮显示<div class="wrapper d">测试条，并在底部显示提示信息：大于等于 900px；当设备宽度小于等于 480px 时，则高亮显示<div class="wrapper a">测试条。演示效果如图 25.1 所示。



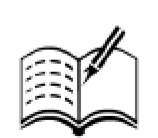
（a）显示屏幕宽度小于等于 600px



（b）显示屏幕宽度介于 600px 和 900px 之间

图 25.1　使用@media 规则





Note



（c）显示屏幕宽度大于等于 900px

图 25.1　使用@media 规则（续）



线 上 阅 读

具体代码解析请扫码学习。

视 频 讲 解

25.2.2　设计响应式版式

本例在页面中设计 3 个栏目：

☑　<div id="main">：主要内容栏目。

☑　<div id="sub">：次要内容栏目。

☑　<div id="sidebar">：侧边栏栏目。

设计当显示屏幕宽度在 999px 以上时，三栏并列显示，预览效果如图 25.2 所示。

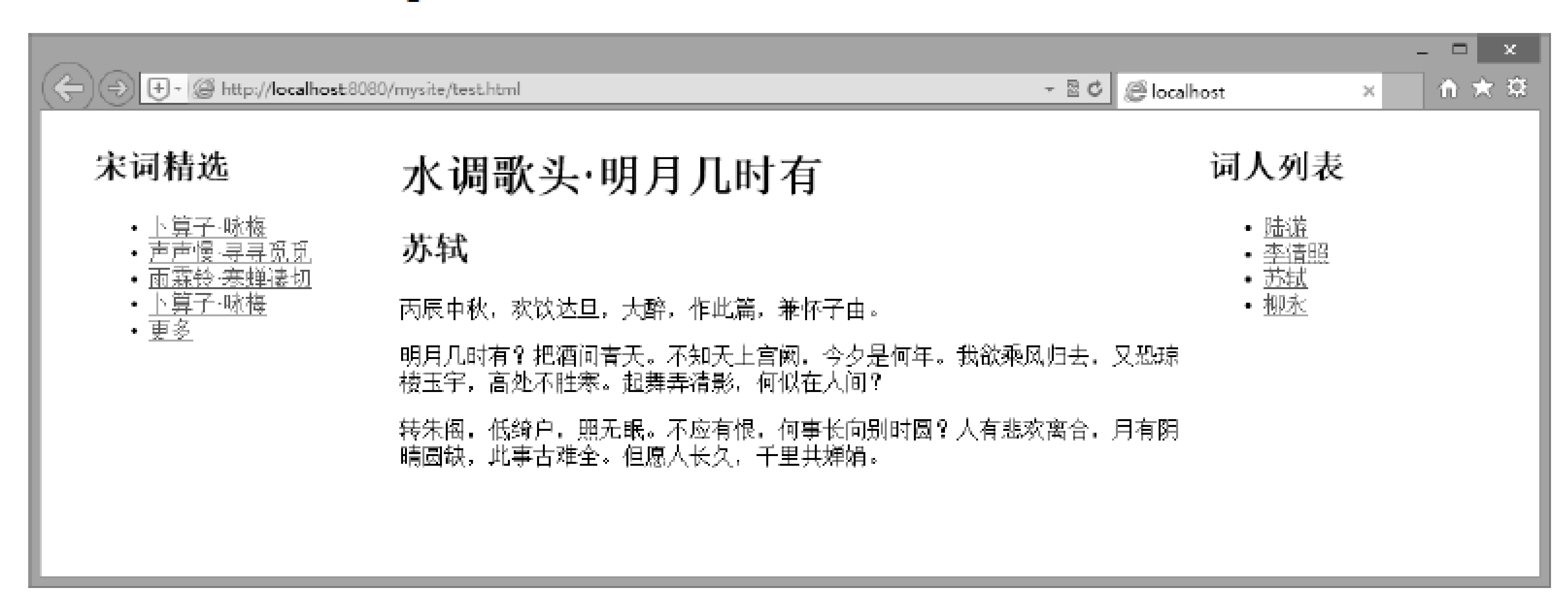


图 25.2　显示屏幕宽度在 999px 以上时页面显示效果

当显示屏幕宽度在 639px 以上、1000px 以下时，两栏显示，预览效果如图 25.3 所示；当显示屏幕宽度在 640px 以下时，三个栏目从上往下堆叠显示，预览效果如图 25.4 所示。

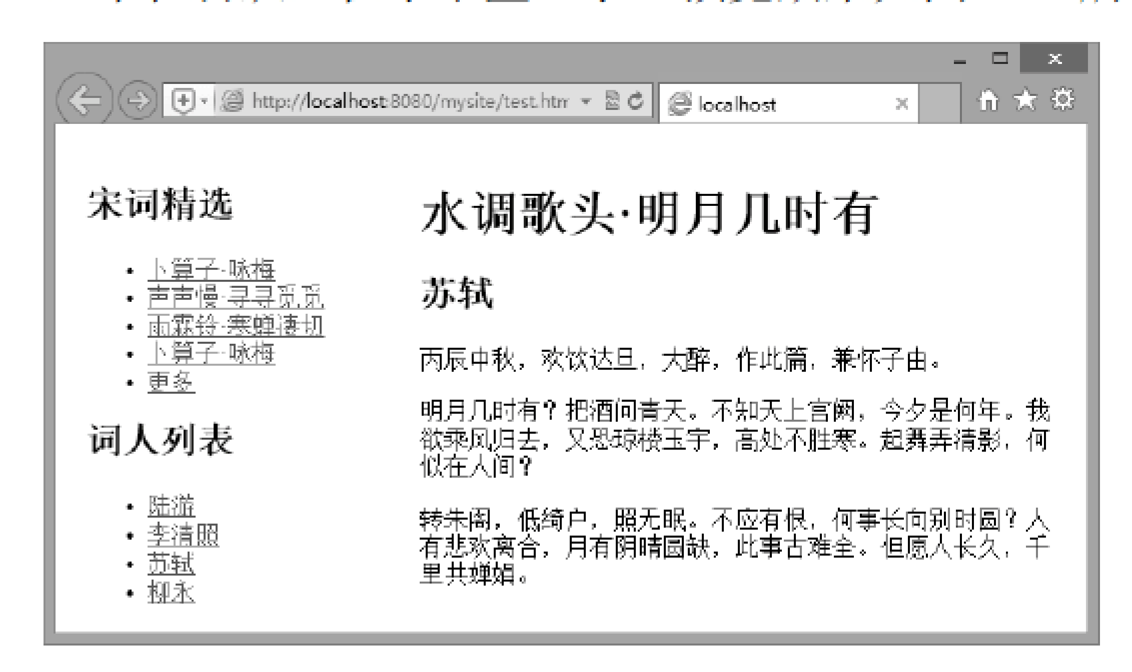


图 25.3　宽度在 639px 以上、1000px 以下时效果







图 25.4　宽度在 640px 以下时效果



线 上 阅 读

具体代码解析请扫码学习。

25.2.3　设计响应式菜单



视 频 讲 解

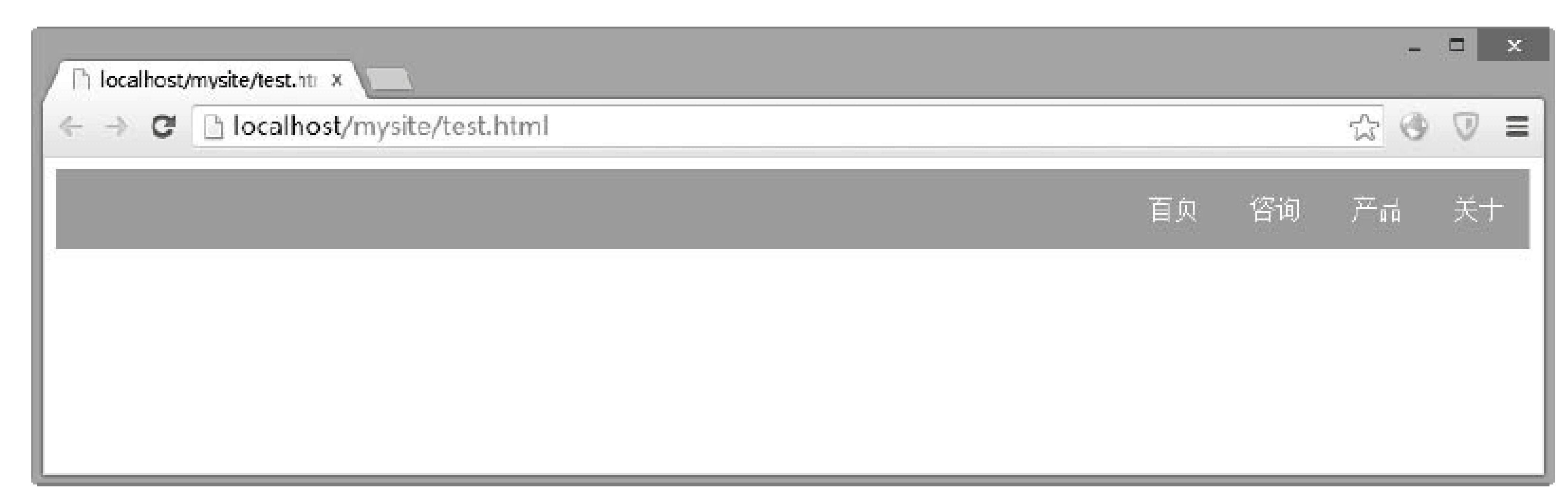
本例设计一个响应式菜单，能够根据设备显示不同的伸缩盒布局效果。在小屏设备上，从上到下显示；在默认状态下，从左到右显示，右对齐盒子；当设备小于 801px 时，设计导航项目分散对齐显示，示例预览效果如图 25.5 所示。



（a）小于 601px 屏幕



（b）介于 600 和 800px 之间屏幕



（c）大于 799px 屏幕

图 25.5　定义伸缩项目居中显示



线 上 阅 读

具体代码解析请扫码学习。



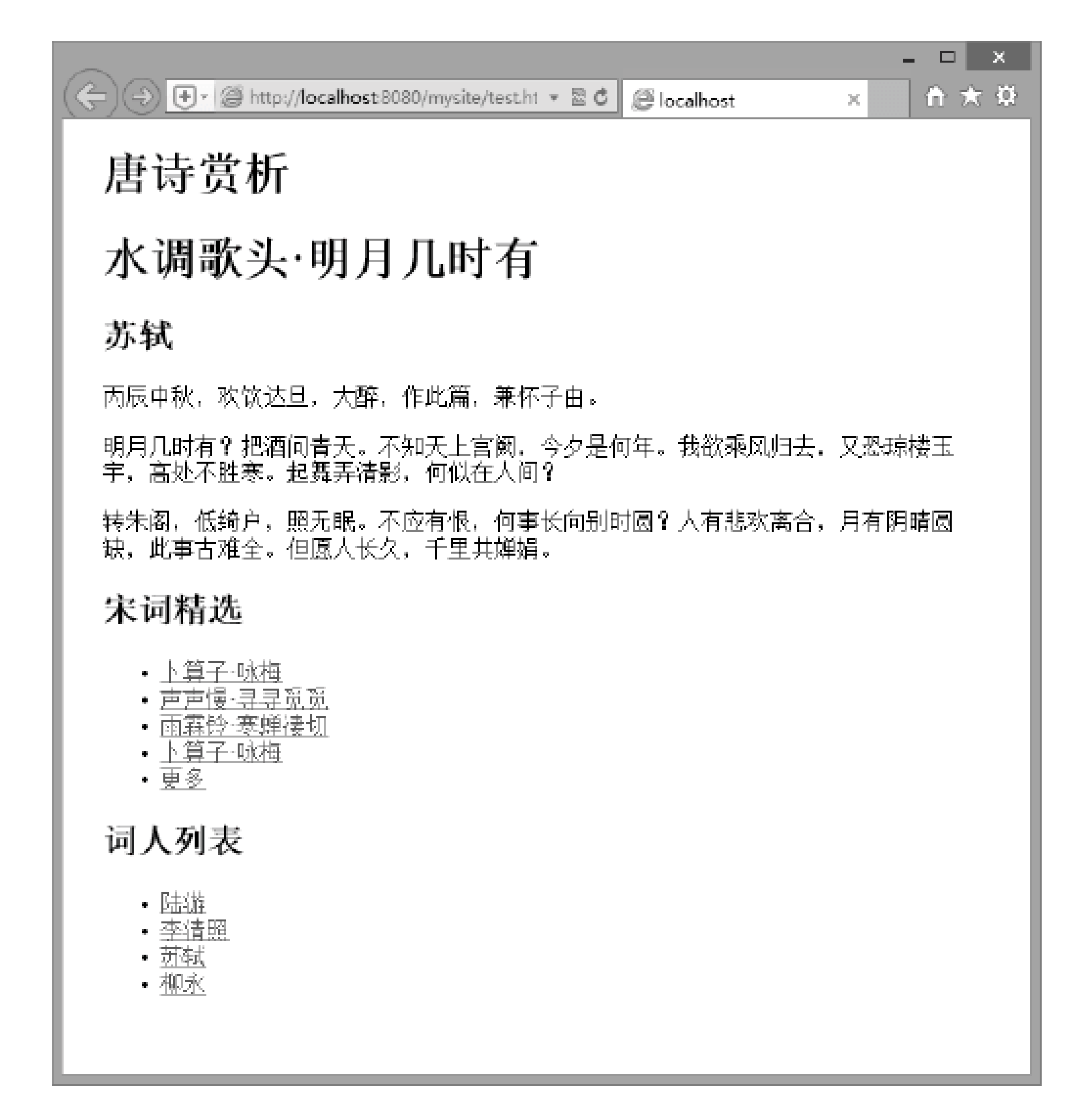


视频讲解

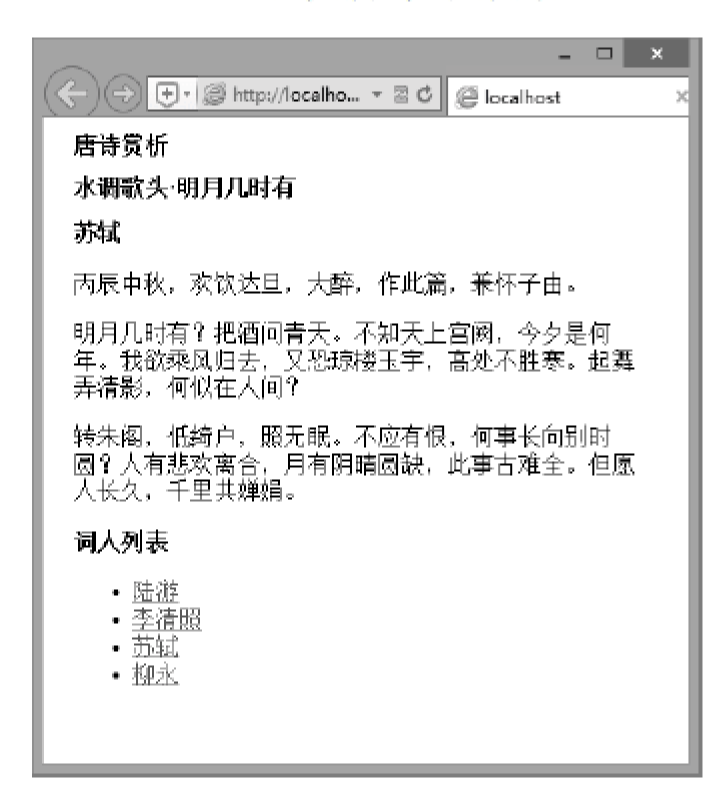
Note

25.2.4　设计自动隐藏布局

本例设计一个响应式页面布局效果，并能根据显示屏幕宽度变化自动隐藏或调整版式显示。演示效果如图 25.6 所示。



（a）平板屏幕下效果



（b）手机屏幕下效果



线上阅读

图 25.6　设计不同宽度下的视图效果

具体操作步骤请扫码学习。



视频讲解

25.2.5　设计自适应手机页面

本例设计页面宽度为 980px，对于桌面屏幕来说，该宽度适用于任何宽于 1024px 的分辨率。通过媒体查询监测宽度小于 980px 的设备，并将页面宽度由固定方式改为液态版式，布局元素的宽度随着浏览器窗口的尺寸变化进行调整。当可视部分的宽度进一步减小到 650px 以下时，主要内容部分的





容器宽度会增大至全屏，而侧边栏将被置于主内容部分的下方，整个页面变为单列布局。演示效果如图 25.7 所示。





图 25.7　在不同宽度下的视图效果

具体操作步骤请扫码学习。



25.3　在 线 练 习

1. 练习响应式网页设计。



2. 复习 CSS3 重要属性，强化训练和应用 CSS3 新功能的能力。

